(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMAGE PROCESSING WITH THE NAVAL POSTGRADUATE SCHOOL INFRARED SEARCH AND TARGET DESIGNATION (NPS-IRSTD) SYSTEM

by

John C. Heiss

June, 1992

Thesis Advisor:	Alfred W. Cooper
Second Reader:	Ron Pieper

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No 0704-0188
1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED		1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; Distribution is unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4 PERFORMING ORGANIZATION REPORT NUMBER(S)		5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School	6b OFFICE SYMBOL (If applicable): 33	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School		
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000		7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000		
8a NAME OF FUNDING/SPONSORING ORGANIZATION: PMS-421 and Naval Postgraduate School Direct funded research	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO	PROJECT NO	TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) IMAGE PROCESSING WITH THE NAVAL POSTGRADUATE SCHOOL INFRARED SEARCH AND TARGET DESIGNATION (NPS-IRSTD) SYSTEM

12 PERSONAL AUTHOR(S) Heiss, John C.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1992, June 17	106

16 SUPPLEMENTARY NOTATION THE VIEWS EXPRESSED IN THIS THESIS ARE THOSE OF THE AUTHOR AND DO NOT REFLECT THE OFFICIAL POLICY OR POSITION OF THE DEPARTMENT OF DEFENSE OR THE U.S. GOVERNMENT.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	IRSTD, Framegrabber, Thermal Imaging, AGA Thermovision

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

An analysis of infrared background scenes generated by the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System, and captured with a DT-2861 frame grabber board, was conducted using a FORTRAN program developed to facilitate image enhancement, clutter suppression and visual target discrimination. The developed FORTRAN program, incorporating the X-arRAY subroutine library to provide access to the DT-2861 memory buffers in the 80386 extended memory, provides access to pre-defined spatial frequency filters for image processing. The program was used to process image data obtained concurrently with the NPS-IRSTD and an AGA 780 Thermovision system operating in the same (3-5μm) waveband. Image histograms and qualitative features of the two image types have been compared. Application of image enhancement and edge detection filters to IRSTD scenes with and without background clutter is considered. Visual target enhancement is observed, together with additional generation of image noise.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL: A.W. Cooper	22b TELEPHONE (Include Area Code): (408) 646-2452	22c OFFICE SYMBOL: PH/CR

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

S/N 0102-LF-014-6603

Image Processing with the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System

by

John C. Heiss
Lieutenant, United States Navy
B.S., United States Naval Academy, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

Author: [illegible]
John C. Heiss

Approved by: Alfred W. Cooper
Alfred W. Cooper, Thesis Advisor

Ron Pieper
Ron Pieper, Second Reader

[illegible]
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

An analysis of infrared background scenes generated by the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System, and captured with a DT-2861 frame grabber board, was conducted using a FORTRAN program developed to facilitate image enhancement, clutter suppression and visual target discrimination. The developed FORTRAN program, incorporating the X-arRAY subroutine library to provide access to the DT-2861 memory buffers in the 80386 extended memory, provides access to pre-defined spatial frequency filters for image processing. The program was used to process image data obtained concurrently with the NPS-IRSTD and an AGA 780 Thermovision system operating in the same (3-5μm) waveband. Image histograms and qualitative features of the two image types have been compared. Application of image enhancement and edge detection filters to IRSTD scenes with and without background clutter is considered. Visual target enhancement is observed, together with additional generation of image noise.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I wish to thank my advisor, Prof. Alfred W. Cooper, for providing me the opportunity to work on a project which directly supports the development and implementation of a shipboard weapons system. I also wish to thank Mr. W.J. Lentz for his support throughout this thesis. His insights provided the additional guidance necessary for successful completion of this project.

This thesis is prepared in conjunction with research sponsored by Naval Sea Systems Command, PMS-421, and supported jointly by PMS-421 and the Naval Postgraduate School direct Funded Research Program.

I. INTRODUCTION

A. BACKGROUND

The development of infrared systems for detection and tracking of airborne targets has long been an interest of the military, with advantages over radar of being a passive sensor and less susceptible to countermeasures [Ref. 1]. The AN/SAR-8 Infrared Search and Target Designation System (IRSTD) is such an infrared system with a primary function of search, track and target designation of all types of antiship missiles, while being insensitive to jamming, radar cross section (RCS) reduction, and anti-ship missile (ASM) guidance mode. The Advanced Demonstration Model (ADM) of the AN/SAR-8 was delivered to the Naval Academic Center for Infrared Technology (NACIT) at the Naval Postgraduate School (NPS) in 1985 to facilitate further research. Following numerous modifications and necessary repairs to the ADM, including bypass of the background normalization unit, the modified ADM or NPS-IRSTD is capable of measuring background radiance signals from a variety of background scenes under varying atmospheric conditions. Evaluation and design of infrared detection systems relies heavily on prior knowledge concerning the characteristics of the background scenes in which they will be employed [Ref. 2].

The primary focus of research conducted using the NPS-IRSTD in the past several years has been the real time display of background scenes generated from the IRSTD, with the development of some processing techniques to enhance image resolution, thereby accentuating distinguishing features of background scenes. This facilitates further research in determining clutter suppression and target discrimination algorithms, as well as development of matched spatial frequency filters to enhance the target detection and tracking capabilities of the AN/SAR-8 [Ref. 3]. A significant development in the real time imaging of the IRSTD was accomplished by Bernier [Ref. 4] through the implementation of a 32-bit assembly language program capable of acquiring and processing data from the IRSTD within the system's 2 second rotation period. This development has enabled the rapid acquisition of data as well as the ability to selectively locate background scenes of interest for further analysis.

B. RESEARCH OBJECTIVES

With an efficient means to acquire background scene information from the NPS-IRSTD already established, the focus of current research is the analysis of background scenes in route to the development of techniques to enhance image resolution, suppress background clutter and improve target discrimination. Additionally, since the IRSTD is an ac coupled infrared system, and therefore not radiometric, comparison of background scenes taken with the IRSTD and a radiometric infrared imaging device, the AGEMA AGA 780 Thermovision system, is performed to establish any correlation between the two intrinsically different infrared systems. Data was acquired with the two systems on 24 September, 1991 to facilitate the scene comparison.

In support of the scene analysis and comparison, a software processing package was developed capable of manipulating background scenes from both the IRSTD and AGA systems. The package includes basic statistical routines, image manipulation functions, and a number of common filtering kernels. Additionally, a slight modification to Bernier's data acquisition routine was made to further enhance the data collection process. The remainder of this work will present an overview of the NPS-IRSTD and AGA 780 Thermovision systems, describe the processing package developed, and conduct an analysis of the scenes acquired with the two systems, with emphasis on techniques employed to enhance the imaging and target discrimination properties of the IRSTD system.

II. SYSTEM OVERVIEWS

The analysis of background scenes acquired with the NPS-IRSTD and comparison with those of the AGA Thermovision is dependent on the imaging properties of each system. The IRSTD system, for example, was not designed as an infrared imaging device, but rather it was optimized for detection of small aspect targets in a variety of flight profiles. The IRSTD and AGA are two intrinsically different infrared devices, the IRSTD being an AC coupled system, and the AGA a radiometric device. It is equally important to realize the differences in data acquisition, as each process invokes some limitations in the ability to compare background scenes. With this in mind, a brief overview of each system is provided including a description of the data acquisition process.

A. NPS-IRSTD

1. System Description

The NPS-IRSTD is a modified ADM of the AN/SAR-8, designed for the passive detection of targets in a full 360° by 10.5° (0.5° below horizon to 10° above) field of view (FOV). It is comprised of a scanner assembly located on the roof of Spanagel Hall at NPS, and associated electronics, including analog-to-digital converters and control panels, located in room 703 of Spanagel Hall. The scanner assembly houses a Schmidt F/1 telescope, two parallel vertical detector arrays comprised of 90 Indium Antimonide (InSb) detectors operating in the 3-5 μm range, a cryogenic cooler, pre-amplifier bandpass assemblies, multiplexing hardware, slip-rings and position-in-rotation signal generating hardware. The detector arrays, termed the lead and lag arrays, are positioned with an approximate 1/2° horizontal separation, and each detector provides a 2.0 by 0.3 mrad FOV. [Ref. 4, 5]

The 90 detectors are connected to six analog multiplexers, whose outputs are sent through a set of slip rings and routed via six analog lines to the six 8-bit analog-to-digital converters. Each detector is sampled 60,000 times per revolution, or approximately once every 0.1 mrad, with a rotation rate of 2.16 seconds, producing a data rate of 5 Megabytes per second for the 180 detectors. In the system's current configuration, however, only the lag array is operational, providing data from 90 of the 180 detectors. Data is then transferred via coaxial cable to room 210 of Spanagel Hall where it is stored on an Ampex HBR 3000i twelve channel tape recorder [Ref. 4].

2. Data Acquisition

Data sent to the HBR 3000i tape recorder can then be processed immediately for real-time analysis, or stored for analysis at a later date. The software developments of Bernier have enabled both the real time analysis of data and an efficient means for post-collection processing. Data from the tape recorder is sent via an interface circuit to the external input port of a DT-2861 Frame Grabber board installed in an 80386 33 MHz computer. The DT-2861 consists of 4 Mbytes of on-board memory and an 8-bit Arithmetic Logic Unit (ALU) providing the capability to capture, manipulate and display up to 16 512x512 8-bit images at a rate of 30 frames per second [Ref.6]. Using Bernier's 32-bit protected mode assembly language program, 184,320 bytes of data is read into a frame buffer on the frame grabber board. The 184,320 bytes corresponds approximately to two 10.5° by 3° images; each detector is sampled 60,000 times per revolution, and 512 pixels is approximately 3°. However, the data is formatted in accordance with the sampling order of the detectors and must be unscrambled prior to image display. The detectors are sampled starting with the second detector from each of the 12 multiplexers (6 for each detector array). The input data is unscrambled in an adjacent frame buffer, resulting in two 3° wide images, separated by a dark zone representing the unprocessed lead array. The two images are offset by approximately 1/2°, representing the separation of the arrays, and are therefore combined to form a 6° image. Since each 2.0 by 0.3 mrad detector of the array is represented by one pixel, the final image is only 90 pixels high and is not proportionally correct. Therefore, the program expands the final image by a factor of 5 in the vertical for a final

512x450 8-bit image representing 10.5° by 6° scene. Images are displayed on a dedicated RGB monitor. All of these processing operations are done on the frame grabber memory buffers, and therefore, upon termination of the program, images are available for further manipulation and storing. A more in depth review of the Bernier's "loadup.exe" program is available in Reference 4.

3. Imaging Properties

The IRSTD was designed to maximize the sensitivity of detection of small angular extent targets at the expense of imaging resolution [Ref.7]. Subtracting the average value of the background by coupling the detector output to a preamplifier circuit which blocks low frequency components of the signal can increase the apparent contrast of the infrared scene. The IRSTD is AC coupled with a low frequency cut-on at approximately 100 Hz. The IRSTD system is therefore not a radiometric device. Rather than measuring the apparent temperature of objects in a scene, the IRSTD detects temperature differences and contrasts in the scene. Hence, gradients are only shown in an IRSTD image if a temperature difference exists through the horizontal scan of the detector array. Thus, background scenes acquired on an overcast day and a clear day would appear the same provided no temperature fluctuation across the scan is present [Ref. 4].

Additionally, IRSTD images are subject to variations due to the responsivity of individual detectors. Engel [Ref. 5] corrected for these difference in his processing routines based on the calibration results of Ayers [Ref.8]. A number of dead detectors were found by Ayers and will appear in an IRSTD image as dark horizontal bands. Corrections for detector responsivity have not been directly targeted in this work, although many of the processing routines implemented correct for the sensitivity differences.

Another phenomenon inherent in the IRSTD system is referred to as droop and undershoot response [Ref. 5]. Droop is a product of the AC coupling scheme employed by the IRSTD and is the result of scanning extended regions of high temperature. Undershoot, on the other hand, occurs when quickly scanning from a high temperature region to a low temperature region and results in a dark or cold region to the right of an object detected by the IRSTD.

B. AGA 780 THERMOVISION

1. System Description

The AGEMA AGA 780 Thermovision is a fourth generation thermal imaging system designed for image recording and analysis. The system consists of a dual scanner, a Black/White monitor chassis for each scanner unit, a color monitor, and an AT computer configured with AGEMA system cards and software. The dual scanner consists of two systems: a shortwave system (SW) using Indium Antimonide (InSb) detectors and operating in the 3-5.6 μm band, and a longwave system (LW) using Mercury Cadmium Telluride (HgCdTe) and operating in the 8-14 μm band [Ref.9]. Each scanner unit consists of an electro-optical scanning mechanism, infrared detector, liquid nitrogen Dewar, and control electronics and preamplifier [Ref.10]. Electro-magnetic energy is focused by an infrared lens into a system of vertical and horizontal rotating prisms to produce a raster pattern. The rotating prisms provide 400 horizontal scan lines per frame, representing four fields interlaced for one frame. However, only 70 of these lines per field, or 280 lines per frame are used as active imaging lines. One frame is produced in 6.25 seconds, corresponding to a scan rate of 25 fields per second. [Ref. 10] The signal is then amplified and processed within the Black/White (B/W) monitor chassis and applied to the display screen. The monitor chassis contains the necessary controls to adjust image brightness, contrast, thermal level and thermal range.

2. Data Acquisition

The amplified analog video signal from the B/W monitor chassis is sent to an AT computer configured with the AGEMA system cards and software. The computer, denoted the Thermal Image Computer (TIC-8000), is used for real time image display and processing, as well as storage of data in binary files. The input signal, converted to an 8-bit digital signal, is displayed on a RGB monitor as a 140x140 pixel image [Ref.11]. The 140 pixel height represents horizontal scan lines from the first and third field of the 280 line interlaced frame. Additionally, each pixel represents a 0.05° field of view (FOV). The installed software supports a wide range of image analysis functions including thermal chopping, image subtraction, statistical calculations including histograms, and some spatial filtering.

3. Imaging Properties

Unlike the IRSTD, the AGA is a radiometric device which measures the scene radiance. Each pixel is assigned a thermal value proportional to the received photon radiation. Temperatures are measured either directly or relative to an external source [Ref. 11]. The 8-bit value representing the radiance is then displayed in a 140x140 image gray scale or color image.

III. IMAGE PROCESSING

The requirement to process and compare images produced from two different infrared systems necessitated a decision concerning the most efficient and effective means by which images could be manipulated, stored and reproduced in a uniform fashion. Both the IRSTD and AGA systems, as currently configured, provide sufficient resources for image processing. However, little or no correlation could be made if the data was not displayed and processed using the same hardware and software tools.

A. HARDWARE CONSIDERATIONS

There were essentially two choices concerning the hardware to use for the desired image processing: the TIC-8000 system incorporated into the AGA imaging system, or the 80386 computer system with the installed DT-2861 frame grabber board used with the IRSTD. The decision to use the 80386 system and frame grabber was an obvious choice based on processing power, speed, and storage capacity. In addition to the DT-2861 frame grabber board with its 4 Mbytes of memory and dedicated Arithmetic Logic Unit (ALU), the 80386 33 Mhz system is also installed with 8 Mbytes of Random Access Memory (RAM), an 80387 math co-processor, a 300 Mbyte hard drive, a 44 Mbyte Bernoulli Disk drive, a tape drive, and a 5¼" High Density floppy disk drive. With each image file stored from the frame grabber occupying 262,656 bytes of memory, the large storage capacity of the IRSTD 80386 system was a major advantage over the TIC-8000 system.

B. SOFTWARE CONSIDERATIONS

Two basic options were available for processing the AGA and IRSTD data; employ a commercial image processing package which was compatible with both the AGA and IRSTD file formats, or develop a custom program to perform all required operations. Several commercial software packages were used by Engel and Bernier to process the IRSTD images. VGAIPS was one such program used by Bernier for image analysis. VGAIPS is an image display program available through a host of computer bulletin boards as a shareware program. It is specifically designed as an infrared imaging program supporting 8-bit binary files from LANDSAT and EOSAT satellites, as well as a number of imaging FLIRS. It provides some filtering functions, histogram display and equalization, and direct look-up table (LUT) specifications. The program is capable of displaying images in gray scale or pseudo-color in standard VGA modes, 640x480 in 16 colors or 320x200 in 256 colors. Any 8-bit image file can be used with VGAIPS provided the size of the file header is known [Ref. 12]. The primary disadvantages of the VGAIPS program are the limited number of processing functions available and the requirement for image data to be adapted to conform with standard display modes. Another processing option was to use the DT-IRIS and IRIS Tutor packages provided with the DT-2861 frame grabber board. DT-IRIS and IRIS Tutor are similar image processing software packages designed specifically for the DT-2861 arithmetic frame grabber board. IRIS Tutor is a self-contained program which accesses all of the available functions of the frame grabber board using command line syntax. DT-IRIS, on the other hand, is a subroutine library image processing package implementing the same functions, while supporting FORTRAN, C and PASCAL high-level languages. Both methods provide easy access to frame buffer operations (addition, subtraction, offsets, multiply, divide), statistical operations (histograms and summation of pixels), logical operations (AND, OR, XOR), convolution with four available filters (Highpass, Lowpass, Laplacian and Vertical Edge detectors), as well as windowing and look-up table specifications [Ref. 13]. Use of the IRIS Tutor limits the processing capabilities to only those operations supported by the command syntax. Development of a custom program using the subroutine libraries of DT-IRIS, such as the FORTRAN routines "IMAGE.FOR" and "DISPLAY.FOR" implemented by Engel and Bernier, offers access to all available frame grabber operations, while offering the flexibility to implement custom functions. However, the DT-IRIS subroutine libraries for FORTRAN are no longer supported by Data Translation Inc. and many functions are incompatible with Microsoft FORTRAN 5.0 and 5.1. The solution to meet all processing requirements was therefore to develop a custom program encapsulating the functions of the Data Translation programs while providing the flexibility to implement additional processing routines as required.

C. SOFTWARE DEVELOPMENTS

1. General

Since a decision was made to process the IRSTD and AGA data using the DT-2861 frame grabber board, the ability to access the frame buffer memory area was essential. However, the frame grabber board and its buffer memory are located in the extended memory area of the computer, requiring a switch to protected mode operation using a DOS extender. This was the strategy employed by Bernier in the real time display of IRSTD data [Ref. 4]. However, a FORTRAN callable library "X-arRAY" from Davis Associates, Inc, providing access up to a gigabyte of extended memory, is a viable alternative to using a DOS extender or 32-bit compiler. Using the "X-arRAY" routines, the memory area of the frame grabber board is easily accessed for transferring image data to and from frame buffers. Additionally, through the use of port calls to the control registers of the DT-2861, the ALU functions which implement basic frame operations such as addition, subtraction, etc., can be easily accessed providing execution of frame processes in 1/30th of a second. This strategy, and initial implementation, was developed by W.J. Lentz, and provides the foundation on which the processing package developed for this research was developed.

2. X-arRAY Library

The X-arRAY library is specifically designed to allocate portions of extended memory dynamically, carry extended memory allocations between job steps, and move data arrays to and from conventional and extended memory [Ref. 14]. Operations requiring protected mode access are performed by the X-arRAY routines and are transparent to the programmer, although a subroutine is available which alerts the user that a protected mode operation has occurred. The library supports array processing routines such as array addition, subtraction, and multiplication as well as integer and floating point unary operations. The library supports arrays of varying data type including real, integer, complex naturally ordered or complex decimated. Of the many callable routines provided in the X-arRAY library, three are particularly useful in accessing the frame grabber memory buffers and manipulating data : BUFXTD, GETXTD, and A2AXTD.

a. *BUFXTD*

This routine allocates fixed memory buffers beyond extended memory to access memory mapped hardware, i.e., the frame grabber board. An address of the fixed buffer location is passed to the routine, as well as the array dimensions and length. The routine returns a "key" which encodes the absolute address in the specified memory area [Ref. 14]. This address key can then be passed within program subroutines, and may be offset to provide access to appropriate segments of the memory area as desired. The start address of the frame grabber memory is "A00000" in the system's current configuration. Since the DT-2861 only provides access to two contiguous frame buffers at a time [Ref. 13], parameters are passed to the BUFXTD routine to encode an address key for a 2 dimensional array consisting of 524,288 bytes or two frame buffers. Using this configuration, access to the start of even numbered frame buffers is accomplished with the "key" parameter, while access to odd numbered buffers is accomplished by offsetting "key" by 262,144 bytes.

b. GETXTD

This routine allocates extended memory that lies within available RAM that is under the control of an extended memory manager (XMS) or Modified LIM [Ref. 14]. As in the BUFXTD routine, this routine returns an address "key" marking the start of the allocation in extended memory. Arrays may be allocated of varying dimensions and size, provided the memory block can lie entirely within the available RAM, 8 Mbytes for the 80386 system as currently configured. Either the Microsoft XMS or Modified LIM method may be specified to allocate the extended memory area, or the programmer may elect to allow the GETXTD routine to determine which allocation method is in use and allocate memory accordingly [Ref. 14].

c. A2AXTD

This routine provides the transfer of data arrays between extended memory or extended and conventional memory using encoded address keys from other X-arRAY routines or local variables. Arrays may be completely or partially transferred as specified by the dimensionality and length of the data array, in conjunction with any offset applied to the address key of the memory location. This routine is implemented extensively in the developed processing program to enable image manipulation in conventional memory; the image data from a desired frame buffer encoded by an associated address key is transferred to a local 2 dimensional (2-D) data array, the 2-D array is manipulated as desired, and the results are transferred back to a frame buffer either in one transfer operation or in several implementations to allow display of results as processing occurs. A similar routine called A2FXTD operates in the same manner, but allows transfer of data from frame buffers to file and vice versa, enabling analysis of stored data.

3. DT-2861 ALU Functions

As described previously, the DT-2861 frame grabber board is equipped with an 8-bit ALU which provides routine processing functions between frame buffers in 1/30th of a second. This high execution speed was the primary advantage of using the DT-IRIS subroutine library or IRIS Tutor program for the processing of images. However, by setting the appropriate bits in the control registers of the frame grabber board, any desired ALU function can be accessed while realizing the same speed of execution.

The DT-2861 functions are controlled and monitored by thirteen 16-bit read/write registers. Register addresses are relative to a user-configurable base address, 592 hexadecimal in the current configuration [Ref. 6]. Access to these registers is conducted via the routines "iport" and "oport" from the PCTOOLS subroutine library [Ref. 15] which allow the reading and setting of input/output (I/O) ports respectively. ALU functions are controlled through manipulation of bits 12-15 of the Video Input Control/Status Register 1 (INSCR1) at address 592 hex. Inputs to the ALU can be from one of four combinations of display feedback, dedicated feedback and A/D input data. By selecting the appropriate feedback and display paths, the function selected by the ALU bits is performed on the selected frame buffers. Selection of the feedback and display paths typically requires manipulation of the Video Input Control/Status Register 2 (INSCR2) and YPAN Register (YPAN). Chapter 3 and Appendix D of Reference 6 thoroughly describe the available ALU functions as well as a bit description of each control/status register for the frame grabber board. Interfaces to the X-arRAY subroutine library are included in the external file "INTERFCE.INC" and listed in Appendix A of this document. Additionally, review of the X-arRAY primitives [Ref. 14] may also be beneficial in understanding specific subroutine operations.

4. Final Product

Incorporating the extended memory routines supported by the X-arRAY library, the DT-2861 ALU functions, and some specialized routines run in conventional memory, a custom processing package written in Microsoft FORTRAN 5.1 was developed to support processing of both IRSTD and AGA image files for analysis. The program provides the required interfaces with the frame grabber memory buffers to enable access to any of the 16 memory buffers. A source code listing of the processing program "PROCESS.FOR" is included as Appendix A for reference. A description of the program capabilities follows.

a. Initialization/Frame Display

The initialization routines consist of the subroutines "POWERUP" and "INITIAL". These subroutines were written by W.J. Lentz and initialize the frame grabber control registers for operation as well as initializing the input and output LUT's. This program uses a 256 monotonically increasing grey scale LUT, although color LUT's can be programmed for use. Upon initialization, frame buffer 0 is displayed and the user can select the desired frame for display. Input of a negative value will display a menu consisting of various frame operations.

b. Frame/File Utilities

Menu items numbered "0" through "3" offer utilities to copy images between frame buffers, load a stored image file to any available frame buffer, or save an image in any buffer to file. The initial implementation of these routines was provided by Lentz, but they have been modified to provide more flexibility. For example, initial implementations of the "frame-to-file" and "file-to-frame" utilities only allowed read and write operations to be performed to frame buffer 0. This was due to the access restriction to the memory area of the frame grabber board. While the DT-IRIS manual states that "Any buffer can be accessed from the bus at any time without restrictions," [Ref. 6], this is only true when incorporating ALU functions. So in a case such as frame-to-frame copies, a procedure which is implemented using the DT-2861 ALU function, access to any frame buffer is available by setting the appropriate bits in the control/status registers. But to enable access to a memory area for read or write operations, only two frame buffers can be accessed at a time as designated by the "BUSBUF" bit in the OUTCSR (bits 9 through 11). Through implementation of a subroutine

"SETBUSBUF" in the process program, the BUSBUF bits are set based on which frame requires access, and an offset of either 0 or 262,144 bytes is applied to the address key specifying the start of the memory allocation. So, to load an image from file to frame buffer 7, for example, the BUSBUF bits are set to "011" to allow access to frames 6 and 7, and an offset of 262,144 bytes is applied to the address key so that the start of the memory location being accessed is the start of frame buffer 7. To enable storage of an image being displayed to file, a similar call to "SETBUSBUF" is prefaced by a call to the subroutine "GETDISBUFF", which determines which frame buffer is being displayed by accessing the "DISBUF" bits of the YPAN register. The current frame is then accessed by setting the correct BUSBUF bits. Once the X-arRAY address key and the frame grabber board memory buffers are initialized, the actual loading and saving of files is accomplished through the X-arRAY routines "A2AXTD" and "A2FXTD" as described previously. Images are saved in IRIS Tutor format [Ref. 13] with a 512 byte header to enable compatibility with the IRIS processing program.

c. Frame Operations

Basic frame operations such as frame addition and subtraction are covered by menu items 3 through 10 in the process program. Frame clears, negation or inverting frames, addition, and subtraction are performed in approximately

1/30th of a second by implementing the appropriate ALU function. The X-arRAY library does support subroutines which perform these functions as well, but take longer to execute (about 2 seconds). The frame offset, frame scale, frame multiplication, and linear frame operations are all implemented using X-arRAY routines. The most unique of these operators is the linear frame operations function. This is an X-arRAY subroutine which performs a linear combination of 2 arrays, or images in this implementation, including offset by a constant (The frame offset routine actually uses this routine as well). This is also one of the more complex routines presented in the program since access to as many as three memory locations could be required, i.e., two different source locations, and a third unique destination. For this reason, the "GETXTD" routine is required to allocate additional memory for intermediate calculations. In short, the linear operations routine performs the function described by $C=(C1*A)+(C2*B)+K$ where "C" is the destination frame, "A" and "B" are the source frames, "C1" and "C2" are real coefficients, and "K" is an integer offset. The source and destination frame buffers may be distinct, equivalent, or overlapping memory areas [Ref. 14]. While elementary in concept, this function allows image contrast adjustment or histogram equalization such as that described by Bernier [Ref. 4] with one function call.

d. Filtering Operations

Selection of this option provides access to the basis for this program, image processing. A secondary menu will appear which prompts the user to select a predefined or user defined filter kernel to process the image. The predefined filters (28 in total) are primarily convolutionary filter kernels common in image processing applications, and are defined in the external files "INTFILT.INC" and "REALFILT.INC". These external files are included with the source code provided as Appendix A. The implementation of the convolution and other routines used in this section were presented in detail in Lindley [Ref. 16] and translated from C source code to FORTRAN. The general operation of these routines consists of the following steps:

- determine which filtering process to implement
- determine the kernel size, allocate memory accordingly, and retrieve the appropriate kernel
- transfer the image to be processed from the frame grabber memory area to conventional memory using "A2AXTD"
- call the appropriate processing routine such as "INTCONV" for integer convolution or "RECONV" for real convolution
- perform the required operation, and in most cases, write processed data to the destination frame one row at a time
- display the final results and release memory allocations

The filters were selected to provide a variety of processing operations, and in some instances, several implementations of the same filter function using different filter kernels. Additionally, the user can create specialized filter kernels of varying sizes as desired. The filtering functions included with DT-IRIS and IRIS Tutor have been duplicated and results from the "PROCESS" program have been validated with the IRIS routines. Speed of execution for the filtering functions is slower than that of the IRIS routines, but acceptable considering the variety of kernels available (only 4 with IRIS Tutor). IRIS Tutor convolutions are described as requiring approximately 3 minutes for execution (512x512 image) [Ref. 13], but take as little as 20 seconds on the 80386 33 MHz machine. The convolutions implemented in the "PROCESS" program have been timed at 25-35 seconds depending on whether the filter kernel is real or integer valued. Other processes such as median filtering or Sobel edge detection which require sorting a "neighborhood" of pixel values or other operations, can take over a minute to complete depending on image extent, "neighborhood" size, and the complexity of the arithmetic operation required. Processes were optimized as well as possible without resorting to assembly language implementation or excessively encrypted source code. Early in the development of the "PROCESS" program, convolution routines were implemented through the direct read of a 9 pixel neighborhood from the image, followed by one pixel write (10 total operations) which required over 40 minutes to execute. Use of the frame grabber control registers to window an appropriate area to be processed could be a possible approach to decreasing processing times. All filtering routines implemented in "PROCESS" allow the user to select a window within the 512x512 image to process, thus decreasing execution times dramatically for small windows. While filtering only a portion of an image is of questionable value in most cases, it is required to process AGA and IRSTD images properly. This will be discussed shortly.

e. *Statistics*

This option calculates a histogram, mean and standard deviation for the displayed image or a user defined portion of the displayed image. The total number of pixels in the defined window, the summation, mean and standard deviation are written to the screen once calculated, but unfortunately, a routine to plot the calculated histogram was not written. As an alternative, the option to write all calculated data to an external file for manipulation with another program is provided. A MATLAB script file was written to import the calculated data, including histogram, compute the data median based on the histogram, and plot the results. A histogram is used to approximate the probability density function of the image and determine the relative brightness and contrast characteristics of the image [Ref. 17].

f. *Image Expansion/Compression*

The final utility provided in the processing program is to expand AGA and IRSTD images or compress IRSTD images. As mentioned earlier, this must be accomplished to obtain a true representation of the filter effects.

(1) AGA Expansion

The AGA image files as saved by the AGA system software are not compatible with the IRIS file format and must be manipulated by software to be properly displayed. An AGA image file (with an IMG file extension) consists of 20,446 bytes, the first 846 of which comprise the file header. The image data follows, but is organized so that the even number lines of the 140x140 pixel image are stored in the first part of the data region, followed by the odd number lines. The first line of the image is therefore located half way through the data region of the file. The data is unscrambled into the center of a 512x512 data area and saved, maintaining IRIS compatibility and preserving all image attributes. A number of utilities were written for the AGA file conversion and are reviewed in Appendix B.

The AGA image can be expanded to a 512x512 image in order to locate particular features. This is accomplished by cropping the AGA image as specified by the user and expanding by a factor of 4. This effectively reduces the FOV of the AGA image from 7° by 7° to approximately 6.4° by 6.4°, but also reduces the image resolution since each pixel is expanded both vertically and horizontally. While this file expansion could be implemented (and initially was) as part of the file conversion process, distinguishing and locating features of the image could be lost. Even more significant, however, is that an expanded AGA image can not be properly filtered with the 3x3 predefined filter kernel with accurate results since the extent of the kernel does not encompass the extent of the expansion process. Therefore, any processing to be accomplished on an AGA image, must be accomplished either by specifying a large kernel to cover the extent of the expansion process or by processing the image in the centered or compressed state. The latter method is recommended. Then by defining the appropriate window to encompass the AGA data for filtering, the execution time for the filtering process can be drastically reduced. The filtered image can then be expanded as desired. The expansion process is performed in less than 2 seconds. An AGA compression routine is not available since the expansion process discards image data and therefore an expanded image can not be restored to its initial state.

(2) IRSTD Compression/Expansion

During the process of acquiring IRSTD data from tape or in real time using Bernier's assembly language routine [Ref. 4], the final image is expanded vertically by a factor of 5 to add visually pleasing proportions to the IRSTD image. As in the case of the expanded AGA image, a 3x3 filter kernel lies within the extent of the expansion process. Therefore, for accurate filtering, the image can be compressed to 90x512, filtered with an appropriate window, and expanded back to a 450x512 state. The compress routine executes in under 2 seconds, most filters can be realized in less than 10 seconds, and expansion executes in approximately 5 seconds. While the expansion of an image usually results in a decrease in image resolution, as is the case with the AGA image expansion process, a reduction in resolution with IRSTD images which have been expanded is not readily apparent. Since IRSTD images have a vertical extent of only 90 pixels, scene features are difficult to identify; therefore, IRSTD images are expanded in order to identify distinguishing features of a scene at the expense of image resolution.

IV. IMAGE ANALYSIS

A. DATA COLLECTION

An experiment was conducted on 24 September, 1991 for the purpose of gathering infrared scene data using the AGA 780 Thermovision shortwave (SW) scanner and the NPS-IRSTD. The goal of the experiment was to collect data such that images obtained with the AGA and IRSTD could be compared and analyzed. Over 100 image files were captured with the AGA, and two AMPEX tapes recorded from the IRSTD. A number of events occurred throughout the course of the experiment including the take-off and landing of several propjets from the Monterey airport.

The selection of suitable images from both the AGA and IRSTD was not a trivial undertaking. While the AGA image files are time and date stamped upon storing, the IRSTD data played back from tape is equipped with no such feature. To select an appropriate portion of the recorded data for viewing, a time approximation was made based on the occurrence of a significant event. Once the corresponding event was found in the AGA image files, the tape counter was noted and labeled with the approximate time. Needless to say, not all significant events seen with the IRSTD were recorded with the AGA. The field of view (FOV) for the two systems also complicated the image selection process. The IRSTD has a vertical FOV of 10.5°, representing 0.5° below the horizon and 10.0° above the horizon. The AGA provides a 7° FOV representing 3.5° above and below the scanner aimpoint. Therefore, unless the AGA scanner has an elevation angle of 3.0° above the horizon, the scenes imaged with the AGA will contain different features than those imaged by the IRSTD for a corresponding sector. Analysis of the AGA images acquired in this experiment indicate that the scanner was focussed at or near the horizon, thus providing significantly more scene features below the horizon than images obtained with the IRSTD.

In order to improve the image selection process for the IRSTD, a slight modification was made to Bernier's assembly language program which loads and unscrambles recorded IRSTD data to the frame grabber board [Ref. 4]. The last processing step in Bernier's program was to copy the final image from frame buffer 0 to frame buffer 2 for display. This occurred for each rotation of the IRSTD, so previous images were lost. By replacing this routine with one which accesses the frame grabber control registers to determine the current display buffer, increment that buffer, and then execute a copy, up to 14 consecutive rotations may be stored from any sector. Further description of the replacement routine is included as Appendix C.

The images used for analysis in the remainder of this section are some of the more closely paired AGA and IRSTD images, but unfortunately represent only a small percentage of the data collected.

B. SCENE ANALYSIS

1. AGA / IRSTD Scene Comparisons

Comparison between AGA and IRSTD scenes is of particular interest since the infrared systems are intrinsically different, the AGA Thermovision being a radiometric device, while the IRSTD is AC coupled and therefore not radiometric. During the discussion of the imaging properties of the IRSTD, several anomalies were presented which adversely effect the representation of IRSTD images. Figure 1 is an annotated IRSTD image depicting some of the unique imaging properties of the IRSTD. Dead or low responsivity detector lines are labeled with the corresponding detector number. Low responsivity detectors tend to partition various features of the IRSTD image such as the non-uniform cloud structure about detector lines 13 through 16. Also quite noticeable is the presence of the dead detector, number 76, depicted by a solid black line. Though not particularly obvious in the figure, a vertical noise pattern is evident in the bottom third of the image. The source for this noise has not been determined, but it is believed to be introduced by the last two multiplexers [Ref. 4]. Also evident is the undershoot characteristic, particularly noticeable in the center of the image. Undershoot, a product of the AC coupled system, is the dark region to the right of the imaged cloud formation. Finally, the large black band at the bottom of the image is not a product of the imaging process, but is a result of the display and reproduction of the image. IRSTD images, in expanded form, consist of 450 lines of data. The memory area of the frame grabber board is designed for a full 512 lines of information. Rather than cropping the memory buffer and only producing images of extent 450x512, full 512x512 images are presented, of which the excess pixels are shown uniformly dark.

Figure 2 presents an AGA and IRSTD image of Herrmann Hall at NPS, taken at approximately the same time with a comparable FOV. While Herrmann Hall is not of particular interest from an imaging or targeting standpoint, it does provide a good base to display the differences in the two imaging systems. The AGA image provides an excellent visual representation of the scene, providing good definition at each topographical interface, i.e., trees to bay, bay to shore, etc. The IRSTD image provides the same definition, although not in a visually pleasing fashion. The IRSTD image does, however, clearly indicate the presence of three antennas on the roof of Herrmann Hall. The same antennae are barely discernable in the AGA image, and in fact, may be indistinguishable in some representations of the same FOV. The IRSTD's ability to detect and display the presence of a small change in contrast is a significant advantage. Also note the definition within the cloud structure depicted in the IRSTD image. This cloud structure is presumably not within the FOV for the AGA scanner, and therefore not shown. Lastly, Figure 2 is a good example of the difficulties experienced in the image selection process. While similar horizontal FOVs are realizable, differences with respect to the vertical FOV make comparative scene analysis difficult. A statistical comparison of the images depicted in Figure 2 is attempted in Figures 3 and 4. Figure 3 consists of histograms for both the AGA and IRSTD images, calculated over the entire image, excluding the unused portion of the 512x512 window in the case of the IRSTD. The AGA image is clearly a high contrast image, with a complex distribution function. The IRSTD, on the other hand is a low contrast image, closely following a Gaussian distribution. While histogram equalization or direct LUT editing will increase the contrast of the IRSTD scenes, the shape of the distribution remains the same. Figure 4 is an attempt to window each image to contain only data from the sky region of the images of Figure 2. While the numerical differences in the case of the IRSTD image using a full and partial window are severe, the overall shape of the distribution remains constant. The partial histogram for the AGA reveals that the distribution within the sky region of the image is largely responsible for the non-Gaussian shape of the distribution function.

The absence of clearly identifiable antennae of the roof of Herrmann Hall as depicted in Figure 2a, but more evident in some comparable AGA scenes, introduced some thought as to the impact of direct and indirect sunlight in an AGA scene. Figure 5a depicts another aspect of Herrmann Hall under what is believed to be indirect sunlight. Figure 5b represents a similar FOV, but taken later in the experiment, under what appears to be more direct sunlight. The antennae are more readily apparent in Figure 5b. Review of the constants recorded in the header portion of the AGA files revealed no apparent deviation in system parameters during the recording of these two images. However, since the detected radiance is a function of the emitted and reflected radiance over a scene, the appearance of the antennae in direct sunlight would indicate an increase in the reflected component of radiance such that some objects are more evident against the background. This effect is evident in many of the AGA images recorded during this experiment, but goes unnoticed in similar IRSTD scenes due to the AC coupling design of the IRSTD system. Consequently, targets with low reflected radiance components and emitted radiance components similar to that of the background, may be indistinguishable using the AGA system.

Figure 6a and 6b are comparable AGA and IRSTD images overlooking Monterey Bay in the vicinity of Lover's Point.

These particular scenes were chosen due to the noticeable cloud structures in each image. The IRSTD image, however, depicts contrast differences within the cloud formation and displays these changes quite noticeably. The AGA image also depicts the changes within the cloud formation, but they are not readily apparent. While this may not seem particularly alarming, a key to successful operation of an infrared system such as the IRSTD, designed for detection and tracking of small target, is the ability to suppress background noise and clutter and detect small angular extent targets. It can not be determined, based on the data acquired during this experiment, whether a target within significant clutter will be detected and imaged with the AGA. Investigation of the clutter suppression abilities of the IRSTD will be discussed shortly. Figure 7 shows the corresponding histograms of the images in Figure 6, calculated over the full extent of the image, excluding the unused portion of the IRSTD window.

2. IRSTD Processing

Determination of suitable techniques for clutter suppression and target discrimination is a significant goal in the IRSTD project, and the emphasis of the processing program presented. It was not possible to display the filtering effects of each of the 28 pre-defined filter kernels on various background scenes in this document. Rather, the effects of a few selected filters applied to interesting scenes are presented with some comments as to the behavior of other filters on the same scene.

As described during the discussion of the "PROCESS" program, filtering of the IRSTD images requires compression of the data to its original state prior to processing. Figures 8 and 9 illustrate this requirement using both a median filter and a Sobel edge detector. Figure 8a is an IRSTD image of an aircraft shortly after take-off from the Monterey airport. The background consists mainly of tree tops and a distant hillside. The aircraft, positioned within the boxed region of the image, is trailed by the characteristic dark region caused by system undershoot. The median filter was employed with the hope of smoothing some of the vertical noise structure in the bottom third of the image as described earlier, and also of suppressing some of the undershoot phenomena characteristic of the IRSTD images. A 3x3 filter kernel was applied to the image in its expanded form, with the results shown in Figure 8b. The overall filtering effect was a slight blurring and smoothing of the image, much like that expected of a lowpass filter. Figure 8c depicts the results when the same filter is applied to the image in its compressed state, and then expanded. Quite noticeable is the absence of the target aircraft in Figure 8c. As will later be shown, this effect is not an inherent property of the median filter when applied to IRSTD images, but its occurrence highlights a significant difficulty in the determination of clutter suppression techniques. If the spatial extent of a target does not sufficiently exceed that of its background, it may be undetected or suppressed by the chosen filtering process. Closer examination of the original aircraft image, Figure 8a, reveals that the aircraft is in a region close to some low responsivity detectors and the frequency content of the area immediately above and below the area is relatively low. The median filter, which finds the median value in a 9x9 neighborhood, therefore suppresses the target.

Figure 9 depicts the same aircraft of Figure 8, only this time a Sobel edge detector is applied to the image, first in its expanded state (Figure 9b), then in its compressed or original state (Figure 9c). There are a number of notable characteristics in these filtered images which are characteristic of many of the edge detectors defined in the "PROCESS" program. First is the detection of low or dead responsivity detectors as horizontal edges. This significantly reduces the ability to discern between targets and imaging-produced noise, although the target of interest does appear to be of slightly higher frequency content. If a thresholding algorithm were employed after this particular edge detection process, an high false alarm rate would surely have occurred. The vertical noise pattern present in the images is also adversely enhanced, thus adding considerably more clutter to the filtered image. Based on these consequences, use of spatial filters which inherently detect horizontal or vertical edges was routinely avoided. However, if a smoothing filter can adequately suppress the adverse image attributes of the IRSTD, or the responsivity and noise problem is eliminated, directionally oriented edge detection may be quite useful.

The next sequence of images is used to test the ability of some of the spatial filters to suppress background clutter and detect a target. Figure 10 depicts a two frame sequence of an aircraft entering a cloud formation. In Figure 10a, the aircraft, positioned within the boxed region, has not yet entered the cloud formation. Figure 10b is a frame captured approximately 4 seconds later with the aircraft in the middle of the cloud formation, positioned within the boxed region of the image. Three distinct filters are applied to these images: a 3x3 median filter, a highpass filter, a Laplacian edge enhancement filter, and finally a combination of the Laplacian edge enhancement applied to the output of the highpass filter. Figure 11a is the filter output from processing Figure 10a, the aircraft before the cloud formation, with a 3x3 median filter. The vertical noise pattern and dead detector line are effectively eliminated, and the effects of the low responsivity detectors are reduced. Even more important is the presence of the target. This particular implementation of the median filter did not suppress the target as was the case in Figure 8c. While this filtering process did little to enhance the target, the general hope was that by effectively eliminating some of the noise generated from the IRSTD imaging process, better results would be achieved after processing this image with some form of edge detector or edge enhancement technique. An appropriate "post-median" filtering combination was not successfully determined. The next filter applied was a highpass filter as shown in Figure 11b. While the processing program has 3 highpass filters to choose from, "HP3" provided the best results. The definition of this kernel can be found in the source listing of "INTFILT.INC" in Appendix A. While implementation of this filter did enhance the representation of the target, the filter also introduced some noise in the image. Nonetheless, the target is readily distinguishable. The third filtering process was a Laplacian edge enhancement technique, shown in Figure 11c. Once again, three kernels are available which implement this technique, but the best performance was achieved through "LAP3". The reproduction of this image is not as flattering as the actual display of the results. The target is readily apparent, with some noise present in the region of the cloud formation. Once again, however, the vertical noise in the bottom portion of the image becomes a significant problem. Application of a thresholding technique to the filtered image may help enhance the target and suppress the vertical noise. The final process applied to Figure 10a was a Laplacian edge enhancement applied to the output of the highpass filter, and is shown as Figure 11d. The reproduction of this image detracts from the actual filtering process, yet it accurately depicts the large amount of clutter resulting from implementation of the highpass filter. The target, however, displays a greater frequency extent than that of the other implemented processes.

The final step was to apply the same filter kernels to Figure 10b, that of the aircraft among the cloud formation. Figure 12a shows the results from the median filter. The target is slightly enhanced in this implementation, but not enough to be able to accurately discriminate a target out of the cloud formation. The highpass filter, Figure 12b, enhances the target within the cloud formation, but not without introducing some additional noise. The Laplacian edge filter, Figure 12c, once again suppresses the clutter enough to detect the presence of a target, but may require some thresholding to ensure accurate detection. Finally, the Laplacian edge enhancement is applied to the output of the highpass filter, Figure 12d. The results are very similar to those of Figure 11d, however the presence of the target is less noticeable due to the significant increase in noise in the vicinity of the target.

3. General Comments

While limited success in clutter suppression and target detection was demonstrated above, it must be mentioned that the aircraft imaged in these scenes were extremely cooperative targets, moving across the IRSTD scan line providing maximum relative motion at very close range. The IRSTD is optimized for detection of small angular extent targets at significantly greater ranges than those realized during this experiment. A target at long range will be well within the 2.0 by 0.3 mrad detector FOV. It is unknown whether a target meeting this size criterion would be of sufficient pixel extent to be imaged and not rejected as a random noise pattern. Additionally, the targets processed in this analysis were all above ground clutter. An infrared search and designation system like the IRSTD would be most beneficial in a nearland/overland environment. However, due to the noise pattern evident throughout the bottom portion of the image and the dead detector approximately in line with the horizon, processing targets of this nature is not feasible.

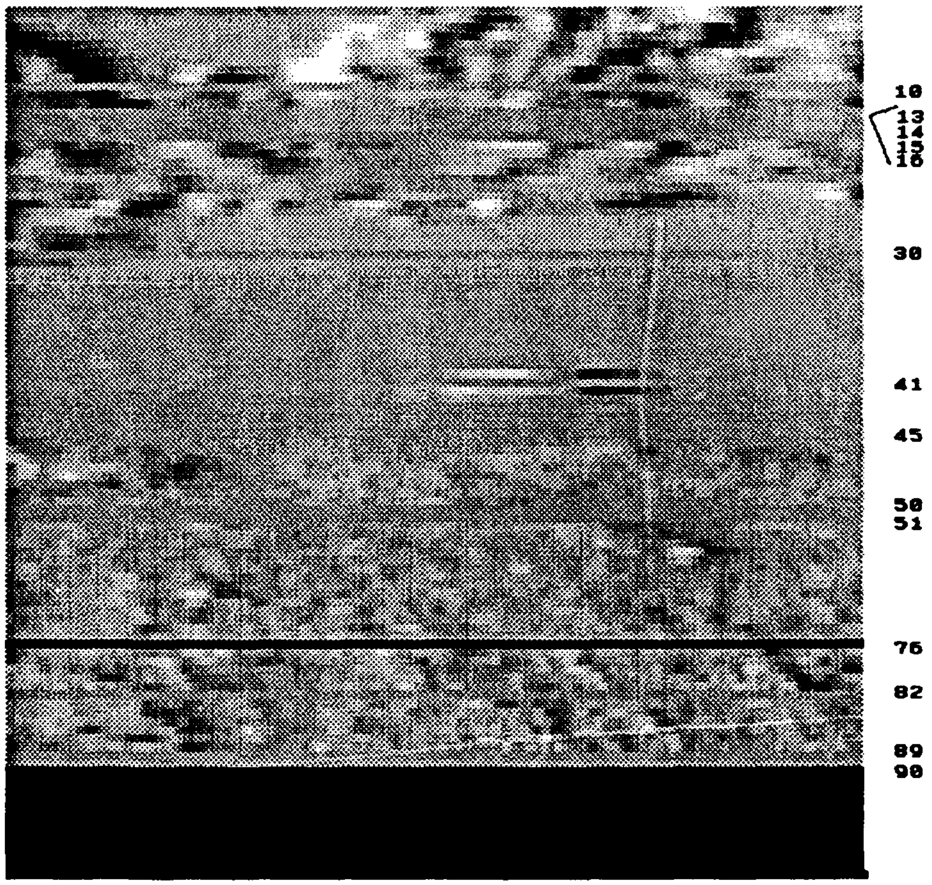


Figure 1. IRSTD image of hillside and cloud cover. Numbers on right indicate dead or low responsivity detectors.

(a)

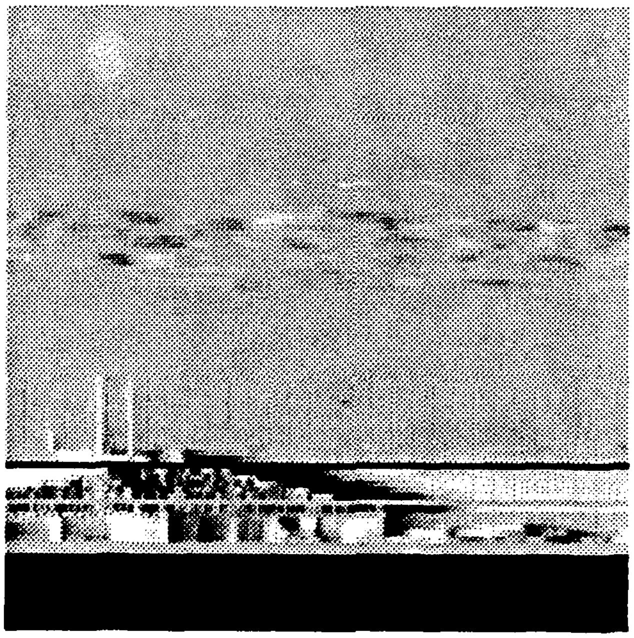
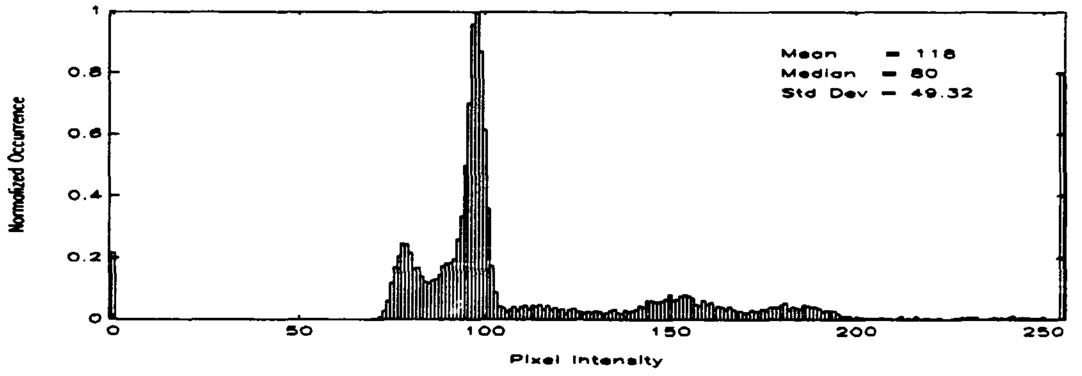
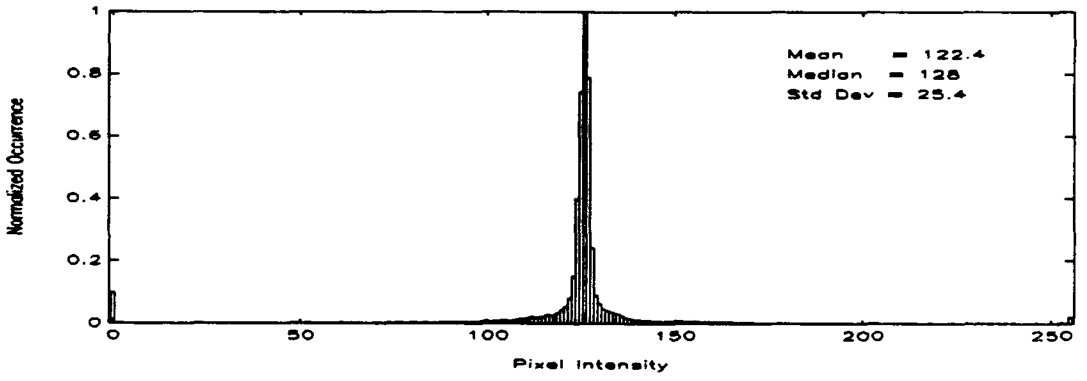

(b)

Figure 2. Comparison of AGA (a), and IRSTD (b) infrared scenes of Herrmann Hall at NPS.

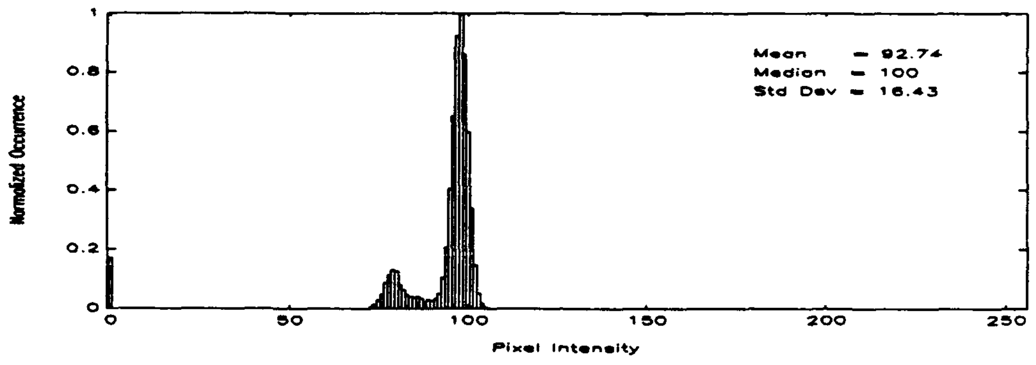


(a)

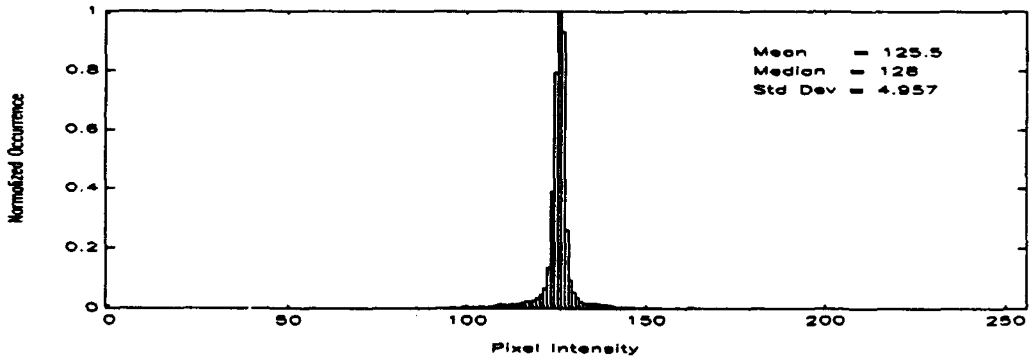


(b)

Figure 3. Histograms of AGA and IRSTD scenes depicted in Figure 2. Histograms over full image. (a) AGA (b) IRSTD

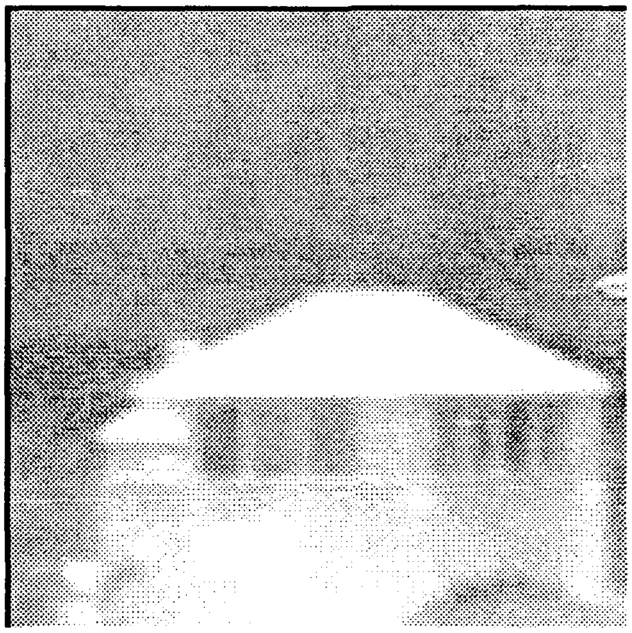


(a)



(b)

Figure 4. Histograms of AGA and IRSTD scenes shown in Figure 2. Histograms calculated over sky portion of image only. (a) AGA (b) IRSTD


(a)

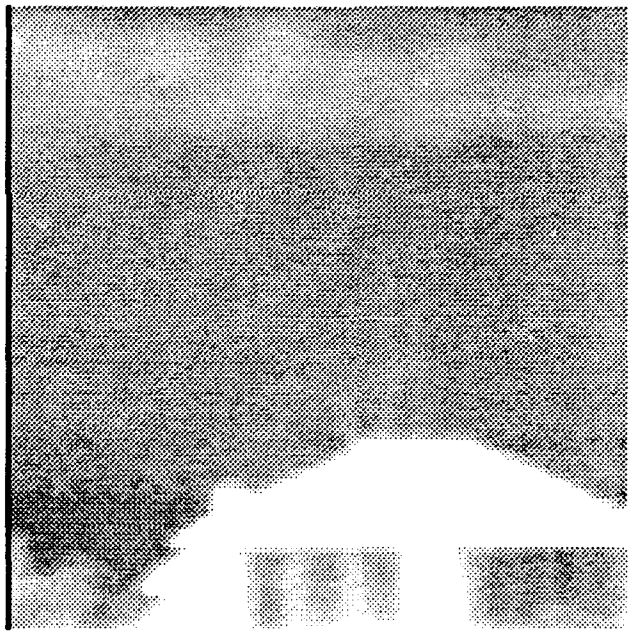
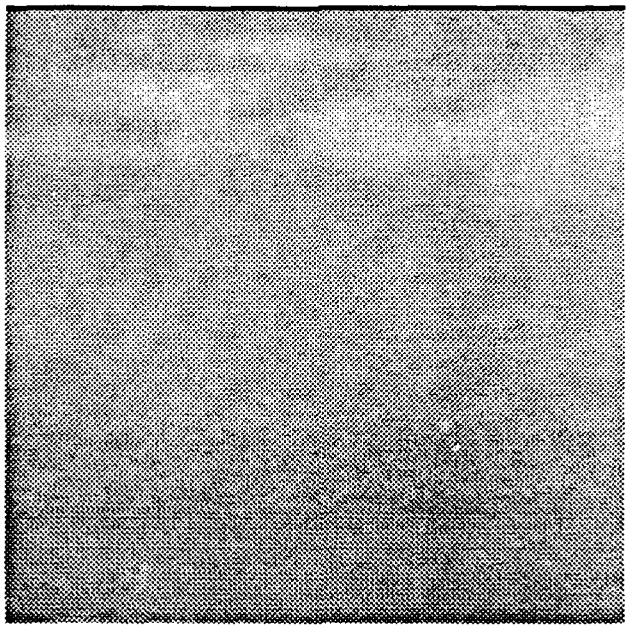

(b)

Figure 5. AGA images of Herrmann Hall taken under different sun conditions. (a) indirect sunlight (b) more direct sunlight


(a)

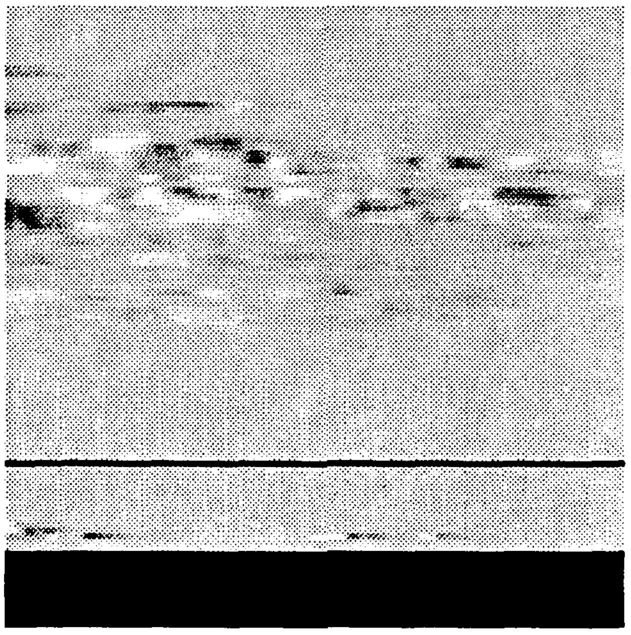
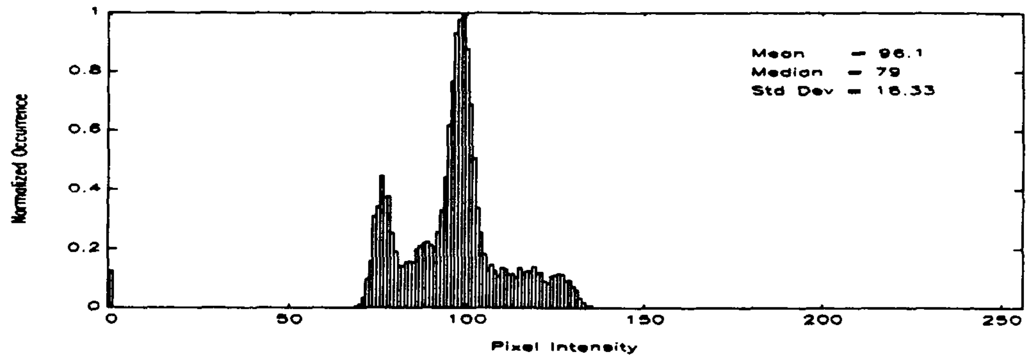
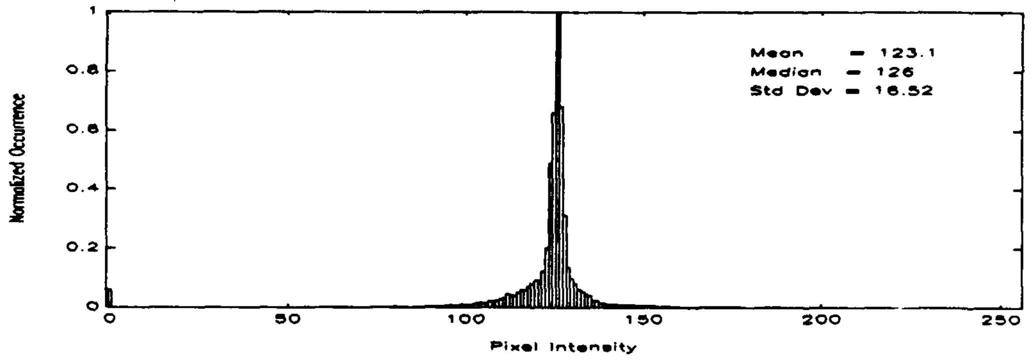

(b)

Figure 6. AGA and IRSTD images overlooking Monterey Bay. (a) AGA (b) IRSTD

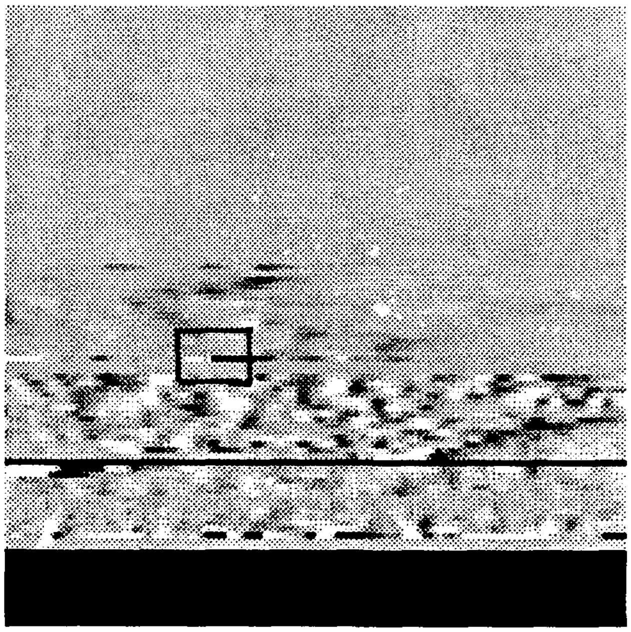


(a)

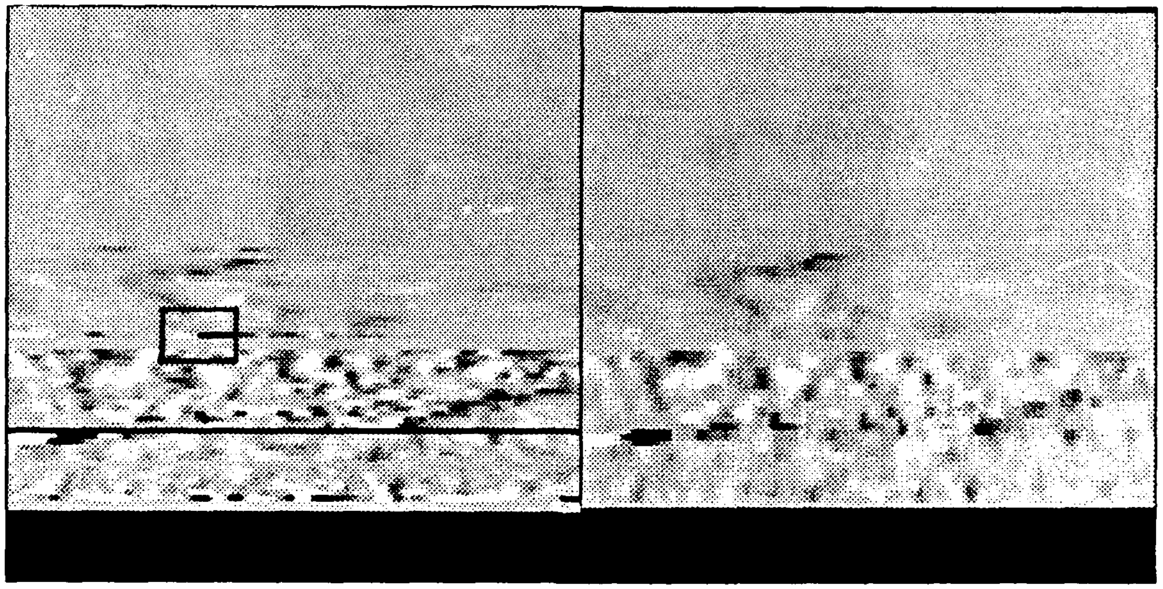


(b)

Figure 7. Histograms of AGA and IRSTD Monterey Bay scenes shown in Figure 6. (a) AGA (b) IRSTD

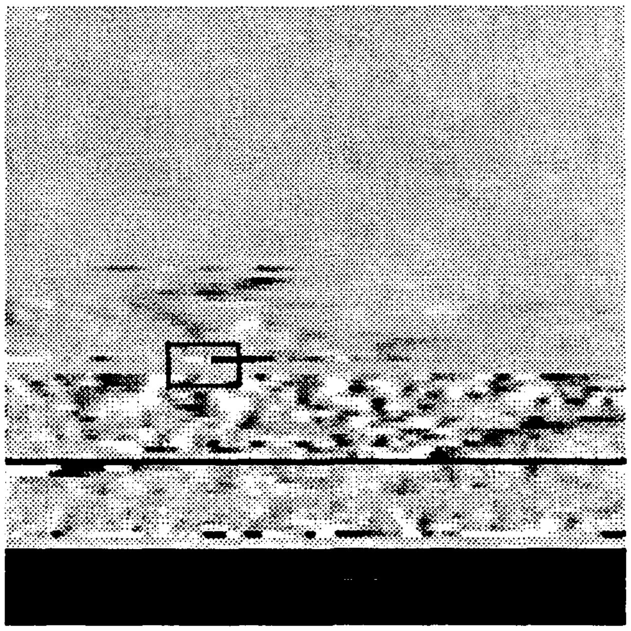


(a)

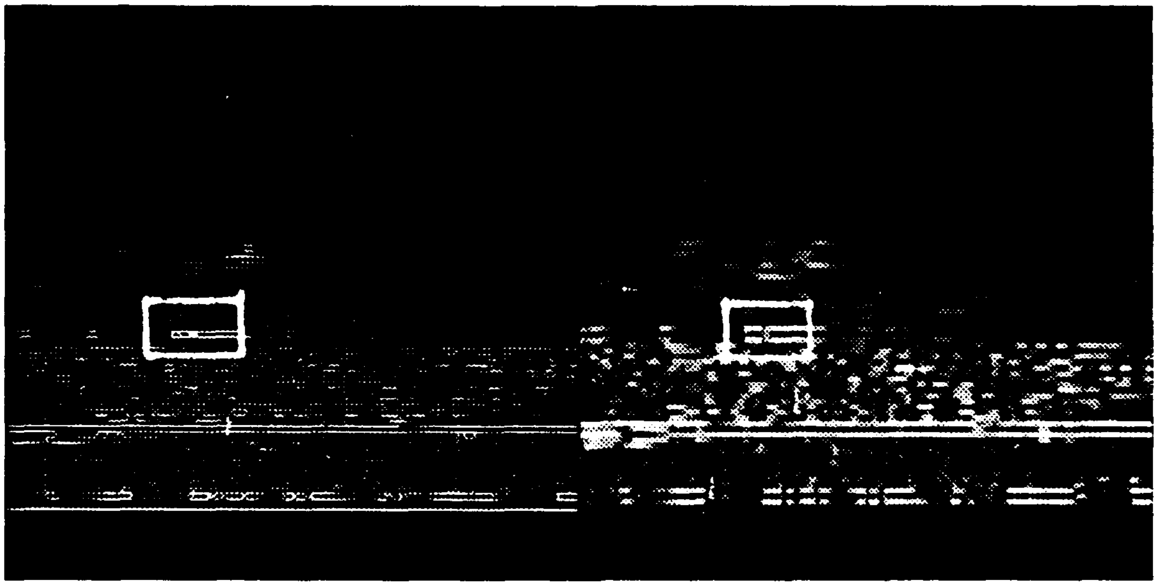


(b) (c)

Figure 8. Effects of filtering IRSTD image in expanded and compressed form. (a) Original image, (b) Original image with median filter, (c) Original compressed, filtered, and expanded

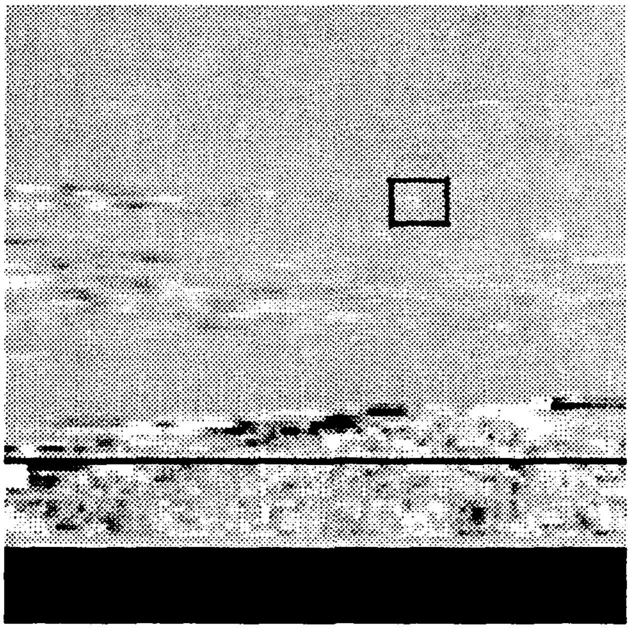


(a)

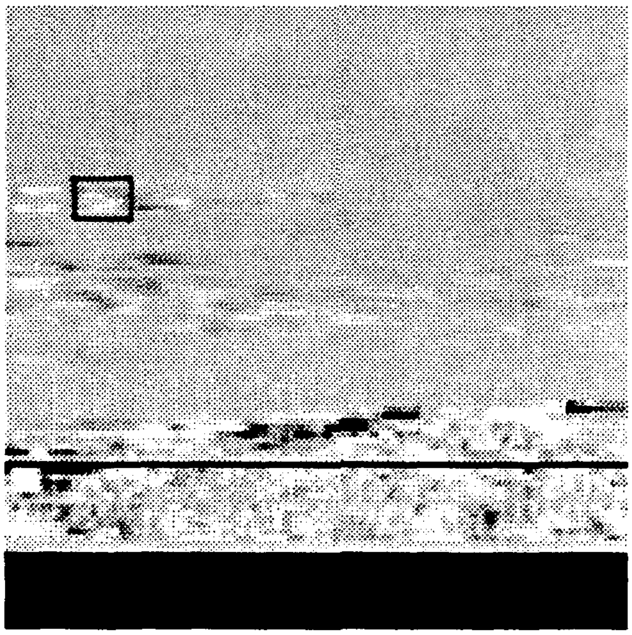


(b) (c)

Figure 9. Effects of filtering IRSTD image in expanded and compressed form. (a) Original image, (b) Original image filtered with Sobel edge detector, (c) Original image compressed, filtered, and expanded



(a)



(b)

Figure 10. IRSTD images of propjet aircraft before (a) and after (b) entering cloud formation.

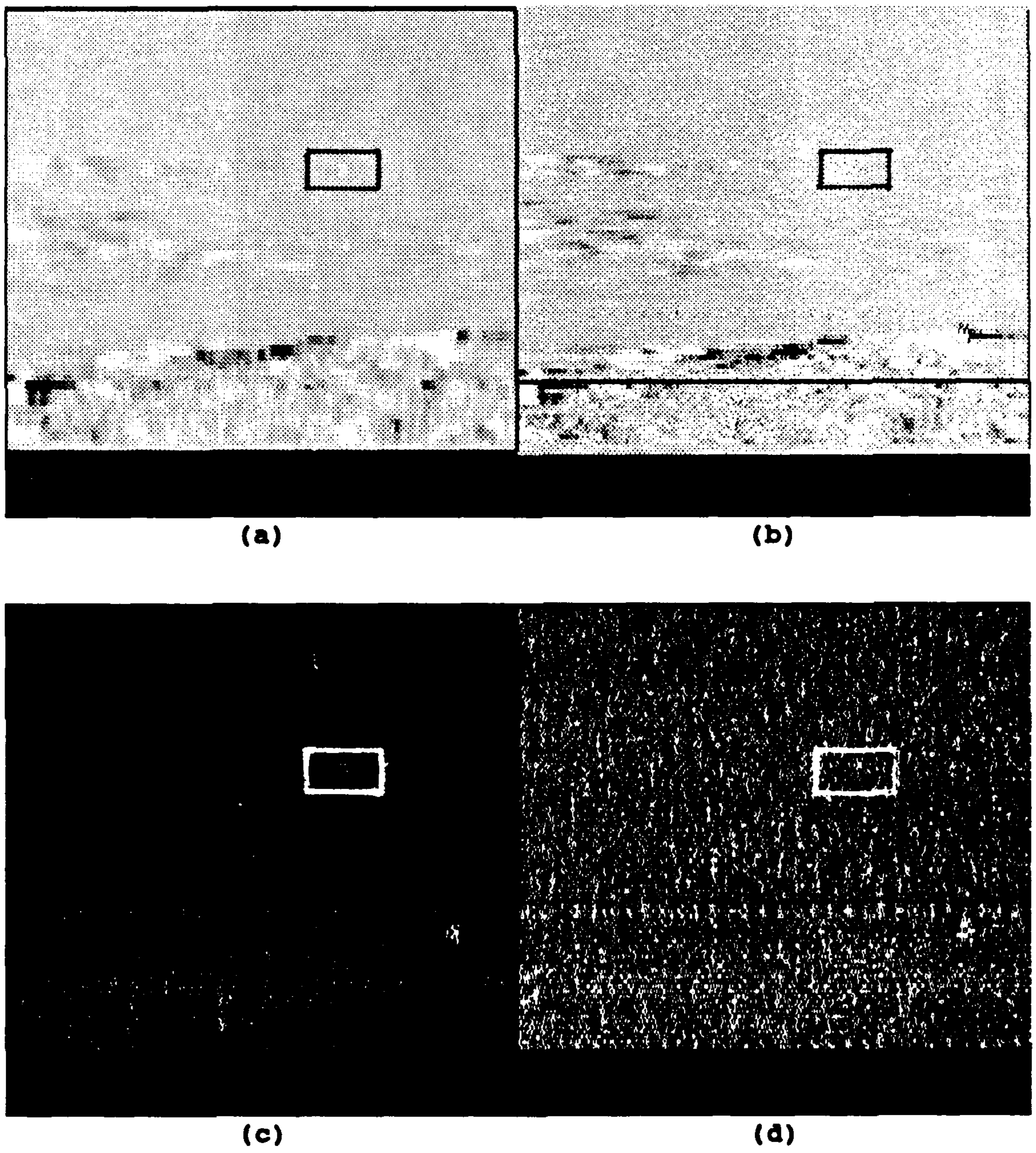
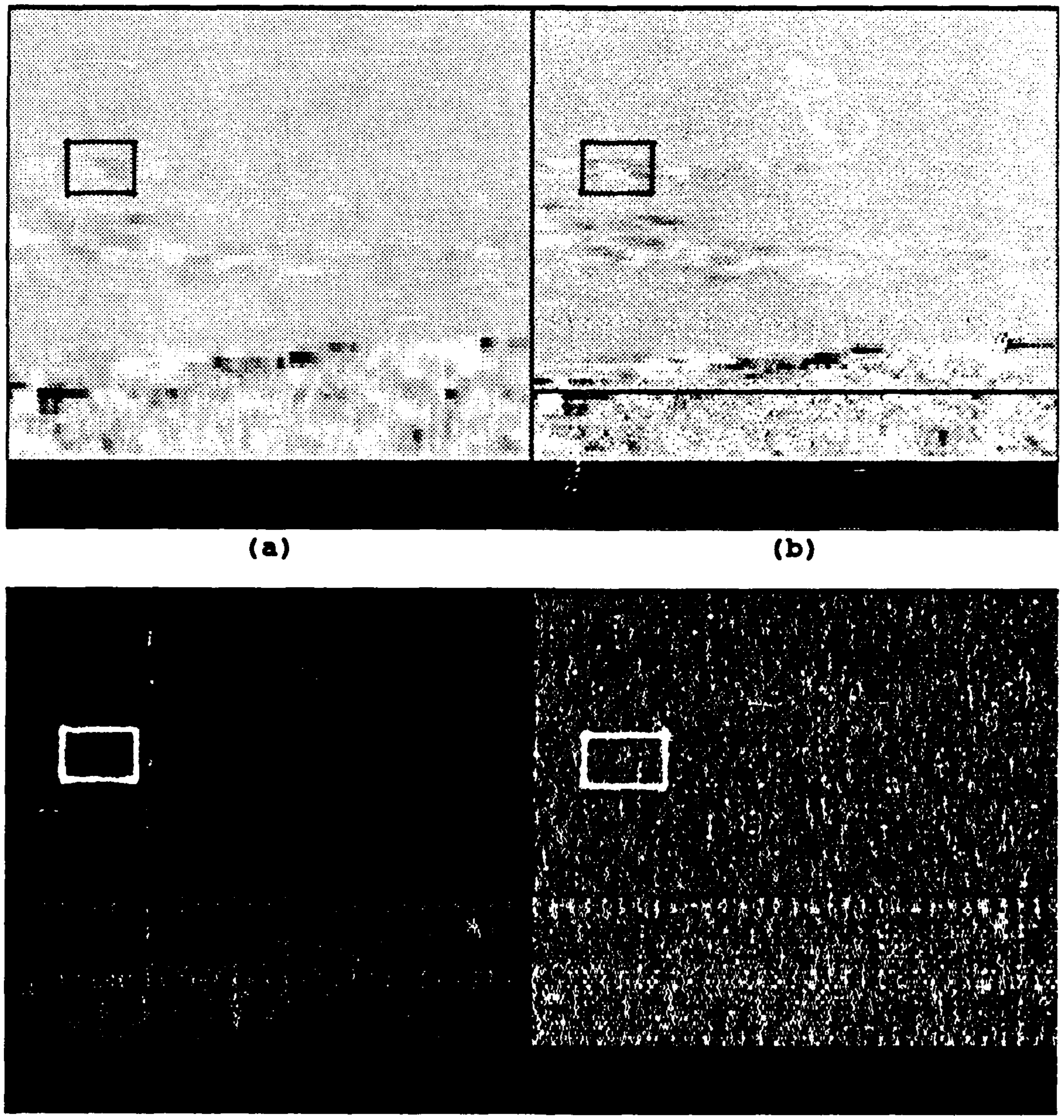


Figure 11. Filtered images of Figure 10a, aircraft before entering cloud formation. (a) Median filter (b) Highpass filter (c) Laplacian edge detector (d) Laplacian edge detector applied to results of highpass (b)


(a) (b)

(c) (d)

Figure 12. Filtered images of Figure 10b, aircraft within cloud formation. (a) Median filter (b) Highpass filter (c) Laplacian edge detector (d) Laplacian edge detector applied to results of highpass (b)

V. CONCLUSIONS AND RECOMMENDATIONS

The recent development of real time imaging with the NPS-IRSTD has provided an efficient means of data collection for the purpose of determining appropriate processing techniques to enhance image resolution, suppress background clutter and improve target discrimination with the IRSTD. The unique imaging properties and data acquisition process required by the IRSTD system necessitated development of a custom processing package which enables access to the DT-2861 frame grabber board memory area and its image processing capabilities. An initial processing package has been developed in support of IRSTD scene analysis, providing some basic statistical functions, image processing routines, and file manipulation and storage. The FORTRAN program uses the extended memory subroutine library X-arRAY to provide the interface into extended memory areas of the 80386 computer, and incorporates use of the DT-2861 control/status registers and onboard ALU to optimize execution of routine functions. The program ensures compatibility of IRSTD image files with IRIS Tutor processing programs and provides sufficient routines to display and process image files obtained with the AGA Thermovision.

Comparison of AGA and IRSTD image files indicates that the IRSTD system is better suited for the detection of small contrast changes within a given region, but does so at the expense of image resolution. Additionally, it could not be determined from the experimental data whether a target could be detected within a clutter region with the AGA Thermovision. Preliminary analysis of IRSTD scenes with appropriate processing techniques indicates that application of omni-directional spatial filters such as the Laplacian edge enhancement filter can provide the most effective means of clutter suppression and target enhancement.

Several recommendations come to mind concerning future research with the IRSTD and comparison with AGA scenes. First and foremost is the conduct of another AGA/IRSTD experiment. Several difficulties were encountered finding suitable scenes for comparison due to conflicting FOV of the two systems, caused by the difference in elevation angle of the AGA scanner. Additionally, an attempt was made to record a sequence of AGA images corresponding to a specific event, much like that acquired by the IRSTD. This was probably over ambitious given the requirements for acquiring and recording an AGA image file. Analysis may be better served by collecting data from a few regions in which it is known that events will occur, such as along the approach line to the Monterey Airport. This would help ensure compatibility of target-rich scenes. Once more comparable scenes are obtained with the AGA and IRSTD, a detailed analysis of scene attributes between the two systems is required.

From an IRSTD perspective, several items require further attention. The question of whether a long range target would have sufficient spatial extent to be displayed by the IRSTD system warrants further investigation. An experiment conducted with the IRSTD and a cooperative air target, preferably with a ranging device, may be able to provide an answer. Also, with the capability to display infrared scenes, it would be interesting to compare scenes obtained with the Background Normalizer Unit (BNU) installed with scenes obtained in the systems current configuration. If nothing else, a determination of the contribution of the BNU overall system performance may be established. Furthermore, elimination of the noise patterns evident in the IRSTD images is essential for proper development of system filter parameters. Efforts to correct or offset with software routines, dead and low responsivity detectors also requires further attention.

A number of modifications to the processing program would certainly enhance research capabilities and program flexibility. The program lacks the necessary tools to perform a detailed statistical analysis of background scenes, including the capability to display ir´ rmation graphically. The X-arRAY subroutine library provides the capabilities to process selected IRSTD images without the requirement of having a frame grabber board. Data could be collected in real-time using established routines for image acquisition and unscrambling, then saved as image files for post-collection processing and analysis. The use of frame grabber memory areas for processing and analysis as implemented in this thesis is not required, but rather provides an efficient means of image processing using available resources. In theory, as long as a computer system has extended memory and a means to display IRSTD formatted images on a VGA screen, an unlimited amount of research can be conducted without using a frame grabber board. The required modifications to the processing program to make this realizable include replacing all references made to the frame grabber memory areas with appropriate memory allocations in extended memory using the X-arRAY routines, and implementing display routines designed to handle IRSTD images on a standard video monitor. Appropriate display routines are already being developed by Bernier for use in this research.

APPENDIX A

The following is the source code for the processing program "PROCESS.for".

```
c   This program implements a host of image processing
c   routines utilizing the memory areas of the DT-2861 frame
c   grabber board. Extended memory areas are accessed via
c   routines from the X-arRAY library and basic frame
c   functions are performed using ALU functions of the
c   framegrabber board
c

$INCLUDE:'interfce.inc'

    integer*4 d,n,lmnt,ihndl,key,kb,ier
    integer*4 mthd,addr,arr1(2),arr2(2)
    integer*2 iret,ier2,ibit,idisp
    integer*1 iy(512,512)
    real*4 y(65536)
    common /y/iy
    equivalence(y(1),iy(1,1))

c       initialize board after computer power down
    call powerup
c       initial clears up LUTS to display gray scale
    call initial
1       print *,'input frame to display or -1 to continue'
read *,idisp
    if(idisp.ge.0) then
        call display(idisp)
        goto 1
    endif
    print *,'Input Desired Selection'
    print *,'        0 frame copies'
    print *,'        1 file to frame'
    print *,'        2 frame to file'
    print *,'        3 clear current frame'
    print *,'        4 invert current frame'
    print *,'        5 add two frames'
    print *,'        6 subtract two frames'
    print *,'        7 offset frame'
    print *,'        8 scale frame'
    print *,'        9 multiply frames'
    print *,'       10 linear 2 frame ops'
    print *,'       11 filtering ops'
    print *,'       12 statistics'
    print *,'       13 expand/compress image'
    read *,ibit

c       open a non contiguous frame buffer
c       (with arr2, gets memory for 2 full frame buffers)
addr=10485760
c       n is the size of the memory in fg for read/write
    n=65536
    arr1(1)=256
    arr1(2)=256
    arr2(1)=512
    arr2(2)=512
    d=2
    lmnt=4
    mthd=-1
    call bufxtd(d,arr2,lmnt,addr,ihndl,key,kb,iret,ier2)
if(ier2.ne.0) then
        print *,'error in bufxtd - ',ier2,iret
        print *,'bufxtd ier,iret,kb = ',ier2,iret,kb
        goto 88
    endif
    select case(ibit)
    case(0)
        call framecopy
    case(1)
        call file2frame(key,arr1,y)
    case(2)
        call frame2file(key,arr1,y)
    case(3)
        call clear
    case(4)
        call negate
    case(5)
        call add
    case(6)
        call subtract
    case(7)
        call offsetfrm(key,arr2)
    case(8)
        call scale(key,arr2)
    case(9)
        call multiply(key,mthd,arr1,arr2,y)
    case(10)
        call linear(key,arr1,arr2,mthd,y)
    case(11)
        call filterselect(key,arr1,iy)
    case(12)
        call stats(key,arr1,iy)
    case(13)
        call adjust(key,arr1,iy)
    case default
        goto 88
    end select
88      call relxtd(ihndl,iret,ier)
c       release the memory above
    if(ier.ne.0) print *,'relxtd error = ',ier,iret
    goto 1
    end

c************************************************************
    subroutine framecopy
c   Routine written by W.J. Lentz
c   copies images between frames

    integer*2 iframes(16),ibuffs(16)
    integer*2 idisp,istop,ibit,word

    print *,'Input frame to display'
    read *,idisp
    call display(idisp)
44      print *,'Input number of frames to copy'
    read *,istop
    if(istop.gt.15) then
        print *,'Error of too many copies'
        goto 44
    endif
    do 55 i=1,istop
54      print *,'input source frame,destination frame'
    read *,iframes(i),ibuffs(i)
    if(iframes(i).lt.0.or.iframes(i).gt.16) goto 54
    if(ibuffs(i).lt.0.or.ibuffs(i).gt.16) goto 54
55      continue

c       Timing call for test of copy rate
    call gettim(ihour,imin,isec,ihn)

    do 66 i=1,istop
    ibit=iframes(i)
    word=ibuffs(i)
    call cmetra(ibit,word)
66      continue

    call gettim(nhour,nmin,nsec,nhn)
    seconds=nsec*100+nhn-isec*100-ihn
    if(istop.gt.0) print *,'hundredths= ',seconds
    call display(idisp)
    return
    end

c************************************************************
    subroutine file2frame(key,arr1,y)
```

```
c    Original implementation W.J.Lentz, modified by Heiss

      integer*2 idisp
      integer*4 key,bytes,ier,arr1(2)
      real*4 y(65536)

      pause
      call rbfile(y)
      print*,'Destination Frame?'
      read*,idisp
      call setbusbuff(idisp,irem)
      call a2axtd(2,arr1,4,locfar(y),key+irem,bytes,ier)
      if(ier.ne.0) then
          print *,'a2axtd error= ',ier
          print *,'bytes,ier= ',bytes,ier
      endif
      call display(idisp)
      return
      end
c***********************************************************
      subroutine frame2file(key,arr1,y)
c    Original implementation W.J.Lentz, modified by Heiss

      integer*2 idisp
      integer*4 key,bytes,ier,arr1(2),irem
      real*4 y(65536)

      call getdispbuff(idisp)
      call setbusbuff(idisp,irem)
      call a2axtd(2,arr1,4,key+irem,locfar(y),bytes,ier)
      if(ier.ne.0) print *,'a2axtd error= ',ier
      call display(idisp)
      pause
      call sbfile(y)
      return
      end
c************************************************************
      SUBROUTINE SBFILE(y)
c    Original implementation W.J.Lentz, modified by Heiss

      character filename*64
      real*4 y(65536)
      integer*1 ihead1(23),ihead2(8),ihead3(461)
      integer*2 dimen
      integer*4 datasize
      character*8  name
      data datasize/262144/
      data dimen/512/
      data name/'DT-IMAGE'/
      data ihead1 /1,4,0,7,9*0,9*0,1/
      data ihead2 /1,1,0,1,1,1,8,1/
      data ihead3 /461*0/

      print *,'Input filename'
      read (*,'(A)') filename
      open (3,file=filename,form='binary')
      write(3) name
      write(3) ihead1,datasize,ihead2,dimen,dimen,
     *         dimen,dimen,ihead3
      write(3) y
      close(3)
      return
      end

c*************************************************************
          SUBROUTINE RBFILE  (y)
c         Original implementation W.J.Lentz, modified Heiss

      character         filename*64
      real*4 y(65536)
      integer*1 idummy(512)
      print *,'Input filename'
      read (*,'(A)') filename
      open (3,file=filename,form='binary')
      read(3) idummy
      read(3) y
      close(3)
      return
      end

c*************************************************************
          subroutine powerup
c         Written by W.J. Lentz
c         Begins initialization of frame grabber board

      integer*2 words(8),word,word2,nport,nport1
      data words/0,0,28800,32639,-12800,1536,2,-2/
      data nport /592/

      call oport(nport,word2)
c         above clears busy
      do I=1,8
          nport1=nport+(I-1)*2
          word=words(I)
          call test(nport,word2)
          call oport(nport1,word)
      end dc
      return
      end

c*************************************************************
      subroutine initial
```

```
c      initial clears display after power down works 2-10-92
       integer*2 nport1,nport,word,ibit,word2
c          setup for data acquisition and copy
       nport=592
       call oport(nport,word2)
c          initialize lut input 0 lut output 0 and busy off

c          initialize outscr with display off
       nport1=nport+4
       call test(nport,word2)
       call oport(nport1,word)
       word=0
c          stop modes
       nport1=nport+2
       call test(nport,word2)
       call oport(nport1,word)

c************************************************************
c          initialize the look up tables
c          set mode ldinlut (mode 2; LDRLUT is mode 3 shift 3)
c
c          Mode ldinlut
       call ldinlut
c
c          clear luts for inlut

       do 55 i=1,256
       call ldinlut
       nport=592

c          p4-19 Index is offset 8 low byte
       nport1=nport+8
       word=i-1
       call test(nport,word2)
       call oport(nport1,word)

c          setup inlut at offset base 10 low byte
       nport1=nport+10
       word=i-1
       call test(nport,word2)
       call oport(nport1,word)
55         continue

       ibit=0
       call rcarry(ibit)
c          carry bit set to 0

       do 57 i=1,256
c          Mode LDRLUT
       call LDRLUT
```

```
c       p4-19 Index is offset 8 low byte
    nport1=nport+8
    word=i-1
    call test(nport,word2)
    call oport(nport1,word)

c       write rlut register
    word=i-1
    nport1=nport+10
    call test(nport,word2)
    call oport(nport1,word)

57      continue
    ibit=1
    call rcarry(ibit)
c       carry bit set to 0

    do 58 i=1,256
c       Mode LDRLUT
    call LDRLUT

c       p4-19 Index is offset 8 low byte
    nport1=nport+8
    word=i-1
    call test(nport,word2)
    call oport(nport1,word)

c       write rlut register
    word=i-1
    nport1=nport+10
    call test(nport,word2)
    call oport(nport1,word)

58      continue


    ibit=0
    call rcarry(ibit)
c       carry bit set to 0

    do 59 j=1,256
    i=j-1

c       Mode LDRLUT
    call LDRLUT

c       p4-19 Index is offset 8 low byte
    nport1=nport+8
    word=i
    call test(nport,word2)
    call oport(nport1,word)

c        red/green registers

    word=i
    word=ishft(word,8)+i
    nport1=nport+12
    call test(nport,word2)
    call oport(nport1,word)

c        write Blue register
    word=i
    nport1=nport+14
    call test(nport,word2)
    call oport(nport1,word)
59       continue

c************************************************************
c        set port 0 for display
    word=-22520
    nport1=nport
    call test(nport,word2)
    call oport(nport1,word)

c        set port 4 for display
    nport1=nport+4
    word=12448
    call test(nport,word2)
    call oport(nport1,word)

c        set port 2 for bit 7 on
    nport1=nport+2
    word=128
    call test(nport,word2)
    call oport(nport1,word)

    nport1=nport+10
    word=1536
    call test(nport,word2)
    call oport(nport1,word)
    return
    end
c************************************************************
    subroutine ldinlut
c        sets mode ldinlut
    integer*2 nport,nport1,word,word2
    word=2
    nport=592
    word=ishft(word,4)
    nport1=nport+2
    call test(nport,word2)
```

```
      call oport(nport1,word)
      return
      end
c************************************************************
      subroutine LDRLUT
c         sets mode ldrlut
      integer*2 nport,nport1,word,word2
      word=3
      nport=592
      word=ishft(word,4)
      nport1=nport+2
      call test(nport,word2)
      call oport(nport1,word)
      return
      end

c************************************************************
      subroutine rcarry(ibit)
c         if ibit is 1, carry bit is set
      integer*2 nport,word,word2,ibit
      if(ibit.eq.0) then
          word=0
      else
          word=256
      endif
      nport=592
      call test(nport,word2)
      call oport(nport,word)
      return
      end

c************************************************************
      subroutine display(ibuff)
c         sets mode display
      integer*2 nport,nport1,word,word2,ibuff
      word=0
      nport=592
      nport1=nport+2
      call test(nport,word2)
      call oport(nport1,word)
c         initial setup
      nport1=nport+4
      word=12448
      call test(nport,word2)
      call oport(nport1,word)
c         select the actual port viewed
      nport1=nport+2
      word=ibuff
      call test(nport,word2)
      call oport(nport1,word)
      nport1=nport+10
```

```
    word=ibuff
    word=ishft(ibuff,12)
    call test(nport,word2)
    call oport(nport1,word)
    return
    end

c************************************************************
        SUBROUTINE test(nport,word2)
c       Written by W.J. Lentz
c       test busy bit for board operations

    integer*2 nport,word2
    j=1
65      call iport(nport,word2)
    j=j+1
    if(j.gt.32000) then
        print *,'32000 test'
        return
    endif
    if(iand(128,word2).ne.0) then
        goto 65
    else
        return
    endif
    end

c************************************************************
    subroutine cmetra(iframe,ibuff)
c       Written by W.J. Lentz
c       subroutine copies any frame to any buffer whether c
displayed or not

    integer*2 nport1,nport,word,ibit,word1,word2
    integer*2 iframe,ibuff

c       port address base for board
    nport=592
    nport1=nport+2
c       the frame number is shifted to bits 8-11
    iframe=ishft(iframe,8)
c       128 sets the feedback to B rather than the pan input
    word1=128+iframe+ibuff
c       are we busy?
    call test(nport,word2)
c       when we are not busy, set up transfer addresses
    call oport(nport1,word1)
    call test(nport,word2)
c       do the copy
    ibit=7
    ibit=2**(ibit)
```

```
      word=ibit+word2
      call oport(nport,word)
      return
      end
c************************************************************
          subroutine setbusbuff(idisp,irem)
c         Sets correct busbuff bits for desired frame

      integer*2 nport,nport1,word,word2,idisp
      integer*4 irem

      word=28832
      nport=592
      nport1=nport+4
      irem=mod(idisp,2)*262144
      if(idisp.gt.1) then
          word=word+(512*int(idisp/2))
          call test(nport,word2)
          call oport(nport1,word)
      else
          call test(nport,word2)
          call oport(nport1,word)
      endif
      return
      end
c************************************************************
      subroutine getdispbuff(idisp)
c         Determines the frame currently being displayed

      integer*2 nport,nport1,word,word1,word2,word3,idisp

      nport=592
      nport1=nport+10
      word=-4096
      call test(nport,word2)
      call iport(nport1,word1)
      word3=iand(word1,word)
      idisp=ishft(word3,-12)
      return
      end

c************************************************************
      subroutine negate
c         Inverts frame being displayed using fremgrabber ALU
      integer*2 nport,nport1,word,word2,idisp
      integer*2 itemp,itemp1

      nport=592
      nport1=nport+2
      call getdispbuff(idisp)
      word=ishft(idisp,8)+idisp
```

```
    call test(nport,word2)
c  save registers
    call iport(nport1,itemp1)
    call iport(nport,itemp)

    call oport(nport1,word)
    word=2184
c  ALU=0000  ALUM=1 ASEL=1  BUSY=1
    call oport(nport,word)
    call display(idisp)

c  restore registers
    call oport(nport1,itemp1)
    call oport(nport,itemp)
    return
    end
c************************************************************
    subroutine clear
c       Clears frame currently being displayed

    integer*2 nport,nport1,word,word2,idisp
    integer*2 itemp,itemp1

    nport=592
    nport1=nport+2
    call getdispbuff(idisp)
    word=ishft(idisp,8)+idisp
    call test(nport,word2)

c  save registers
    call iport(nport1,itemp1)
    call iport(nport,itemp)

    call oport(nport1,word)
    word=12424
c  ALU=0011  ASEL=1  ALUM=1 BUSY=1
    call oport(nport,word)
    call display(idisp)

c  restore registers
    call oport(nport1,itemp1)
    call oport(nport,itemp)
    return
    end
c************************************************************
    subroutine add
c       Adds two frames using ALU with No CARRY

    integer*2 nport,nport1,word,word2,idisp
    integer*2 itemp,itemp1,ioper
    nport=592
    nport1=nport+2
    call getdispbuff(idisp)
    call test(nport,word2)
    print*,'Input Frame to ADD to current display'
    read*,ioper
    word=ishft(ioper,8)+idisp
c  save registers
    call iport(nport1,itemp1)
    call iport(nport,itemp)

    call oport(nport1,word)
    word=-26496
c  ALU=1001  ASEL=1  ALUM=0 BUSY=1
    call oport(nport,word)
    call display(idisp)

c  restore registers
    call oport(nport1,itemp1)
    call oport(nport,itemp)
    return
    end
c***********************************************************
    subroutine subtract
c       Performs subtraction of any two frames using ALU

    integer*2 nport,nport1,nport2,word,word1,word2,idisp
integer*2 itemp,itemp1,ioper

    nport=592
    nport1=nport+2
    nport2=nport+10
    call getdispbuff(idisp)
    call test(nport,word2)
    print*,'Input Frame to SUBTRACT from current display'
read*,ioper
    word=ishft(idisp,8)+idisp
    word1=ishft(ioper,12)
c  save registers
    call iport(nport1,itemp1)
    call iport(nport,itemp)

    call oport(nport1,word)
    call oport(nport2,word1)
    word=26768
c  ALU=0110  ASEL=1  ALUM=0  CARRYIN=1 BUSY=1
    call oport(nport,word)
    call display(idisp)

c  restore registers
    call oport(nport1,itemp1)
```

```
      call oport(nport,itemp)
      return
      end

c*************************************************************
      subroutine linear(key,arr1,arr2,mthd,y)
c           Implements a linear combination of arrays

      integer*2 iret,ier2,idisp
      integer*4 ihndl1,key,key2,kb,bytes,ier
      integer*4 mthd,arr1(2),arr2(2),K,irem
      real*4 y(65536),C1,C2

      print *, 'Input parameters C1, C2, K for the equation '
      print *, '       F = (C1*Fx) + (C2*Fy) + K '
      read *,C1,C2,K
      print*,'Frame Fx ?'
      read*,idisp
      call setbusbuff(idisp,irem)
      call a2axtd(2,arr1,4,key+irem,locfar(y),bytes,ier)
      print*,'Frame Fy ?'
      read*,idisp
      call setbusbuff(idisp,irem)
      call getxtd(2,arr1,4,mthd,ihndl1,key2,kb,iret,ier2)
      if(ier2.ne.0) then
          print *,'error in getxtd - ',ier2
          call relxtd(ihndl1,iret,ier)
          if(ier.ne.0) print *,'relxtd error = ',ier,iret else
          call a2axtd(2,arr1,4,key+irem,key2,bytes,ier)
          call getdispbuff(idisp)
          call setbusbuff(idisp,irem)
          print*,'Result will be put in current frame',idisp
call ilnlum(2,arr2,C1,locfar(y),C2,key2,K,key+irem,ier)
if(ier.ne.0)print *,'ilnlum error=',ier
          call relxtd(ihndl1,iret,ier)
          if(ier.ne.0) print *,'relxtd error = ',ier,iret
endif
      return
      end

c*************************************************************
      subroutine scale(key,arr2)
c           Multiplies a frame by a real number

      integer*2 idisp
      integer*4 irem,arr2(2),ier,key
      real*4    xscale

      call getdispbuff(idisp)
      print*,'Input scale value '
      read*,xscale
      call setbusbuff(idisp,irem)
      call ismlum(2,arr2,key+irem,xscale,ier)
      if(ier.ne.0)print *,'ismlum ier= ',ier,xscale
      call display(idisp)
      return
      end

c*************************************************************
      subroutine offsetfrm(key,arr2)
c         Adds a constant to all pixels of a frame
      integer*4 arr2(2),irem,K,ier
      integer*2 idisp
      real*4  C1,C2

      print *, 'Input desired offset '
      read *,K
      C1=1.0
      C2=0.0
      call getdispbuff(idisp)
      call setbusbuff(idisp,irem)
      call ilnlum(2,arr2,C1,key+irem,C2,
      key+irem,K,key+irem,ier)
      if(ier.ne.0)print *,'ilnlum error=',ier
      call display(idisp)
      return
      end

c*************************************************************
      subroutine multiply(key,mthd,arr1,arr2,y)
c         Multiplies two frames

      integer*2 iret,ier2,idisp
      integer*4 ihndl1,key,key2,kb,bytes,ier
      integer*4 mthd,arr1(2),arr2(2),irem
      real*4 y(65536)

      print*,'Frame #1 ?'
      read*,idisp
      call setbusbuff(idisp,irem)
      call a2axtd(2,arr1,4,key+irem,locfar(y),bytes,ier)
      print*,'Frame #2 ?'
      read*,idisp
      call setbusbuff(idisp,irem)
      call getxtd(2,arr1,4,mthd,ihndl1,key2,kb,iret,ier2)
      if(ier2.ne.0) then
          print *,'error in getxtd - ',ier2
          call relxtd(ihndl1,iret,ier)
          if(ier.ne.0) print *,'relxtd error = ',ier,iret else
          call a2axtd(2,arr1,4,key+irem,key2,bytes,ier)
          call getdispbuff(idisp)
          call setbusbuff(idisp,irem)
```

```
        print*,'Result will be put in current frame',idisp
call implum(2,arr2,locfar(y),key2,key+irem,ier)
if(ier.ne.0)print *,'implum error=',ier
        call relxtd(ihndl1,iret,ier)
        if(ier.ne.0) print *,'relxtd error = ',ier,iret
endif
    return
    end

c************************************************************
    subroutine intconv(key,arr1,iy,
   *kernelno,kerrow,kercol,offcol,offrow,rowstart,rowextent,
   *     colstart,colextent,idisp,irem,buffsize)
c       Implements integer convolution

    integer*2 error,error1,idisp,time
    integer*4 key,ier,arr1(2),bytes,irem,ndx
    integer*2 buff[ALLOCATABLE](:), kernel[ALLOCATABLE](:)
integer*2 rowextent,rowstart,isum,colextent,colstart
integer*1 buffsize,kernelno,iy(512,512),result(512)
    integer*1 kerrow,kercol,offcol,offrow,pix

    if(kernelno.ne.0) then
    ALLOCATE (buff(buffsize),stat=error)
    ALLOCATE (kernel(buffsize),stat=error1)
    call igetfilter(kernelno,kernel,buffsize)
    print*,'This will take a few minutes'
    call gettim(ihour,imin,isec,ihn)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(iy) =',ier
    call setbusbuff(idisp,irem)
    do I=rowstart,rowextent
    do J=colstart,colextent
        pix=1
        isum=0
        do M=0,kercol-1
        do N=0,kerrow-1
            buff(pix)=iy(J+M,I+N)
            if(buff(pix).lt.0)buff(pix)=256+buff(pix)
            isum=isum+(buff(pix)*kernel(pix))
            pix=pix+1
        end do
    end do
    if((kernelno.lt.4).or.(kernelno.gt.6)) then
        if(isum.gt.255)isum=255
        if(isum.lt.0)isum=0
    else
        isum=abs(isum)
    endif
    result(J+offcol)=isum
    ndx=(512*(I+offrow-1)+offcol)
    end do
    call a2axtd(1,128,4,locfar(result),key+irem+ndx,
bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(result) =',ier
    end do
    call gettim(nhour,nmin,nsec,nhn)
    call display(idisp)
    time=(nmin-imin)*60 + (nsec-isec)
    print*,time,' secs'
    DEALLOCATE (buff, Stat=error)
    DEALLOCATE (kernel, Stat=error1)
    endif
    return
    end

c************************************************************
    subroutine reconv(key,arr1,iy,
   *kernelno,kerrow,kercol,offcol,offrow,rowstart,rowextent,
    *     colstart,colextent,idisp,irem,buffsize)
c       Implements real convolution

    integer*2 error,error1,idisp,time
    integer*4 key,ier,arr1(2),bytes,irem,ndx
    integer*2 rowextent,rowstart,colextent,colstart
    integer*1 buffsize,kernelno,iy(512,512)
    integer*1 kerrow,kercol,offcol,offrow,pix,result(512)
real*4 rsum
    real*4 buff[ALLOCATABLE](:),kernel[ALLOCATABLE](:)
    if(kernelno.ne.0) then
    ALLOCATE (buff(buffsize),stat=error)
    ALLOCATE (kernel(buffsize),stat=error1)
    call rgetfilter(kernelno,kernel,buffsize)
    print*,'This will take a few minutes'
    call gettim(ihour,imin,isec,ihn)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(iy) =',ier
    call setbusbuff(idisp,irem)
    do I=rowstart,rowextent
    do J=colstart,colextent
        pix=1
        rsum=0.0
        do M=0,kercol-1
        do N=0,kerrow-1
            buff(pix)=iy(J+M,I+N)
            if(buff(pix).lt.0.0)buff(pix)=256.0+buff(pix)
            rsum=rsum+(buff(pix)*kernel(pix))
            pix=pix+1
        end do
    end do
    if(rsum.gt.255.0)rsum=255.0
    if(rsum.lt.0.0)rsum=0.0
```

```
        result(J+offcol)=rsum
        end do
        ndx=(512*(I+offrow-1)+offcol)
        call a2axtd(1,128,4,locfar(result),key+irem+ndx,
    bytes,ier)
        if(ier.ne.0)print*,'a2axtd error(result) =',ier
        end do
        call gettim(nhour,nmin,nsec,nhn)
        call display(idisp)
        time=(nmin-imin)*60 + (nsec-isec)
        print*,time,' secs'
        DEALLOCATE (buff, Stat=error)
        DEALLOCATE (kernel, Stat=error1)
        endif
        return
        end

c************************************************************
        subroutine igetfilter(kernelno,kernel,buffsize)
c           Sets the approriate integer kernel

        integer*1 kernelno,buffsize
        integer*2 kernel(*)
$INCLUDE:'intfilt.inc'

        select case(kernelno)
        case(1)
            do I=1,buffsize
                kernel(I)=HP1(I)
            end do
        case(2)
            do I=1,buffsize
                kernel(I)=HP2(I)
            end do
        case(3)
            do I=1,buffsize
                kernel(I)=HP3(I)
            end do
        case(7)
            do I=1,buffsize
                kernel(I)=LE1(I)
            end do
        case(8)
            do I=1,buffsize
                kernel(I)=LE2(I)
            end do
        case(9)
            do I=1,buffsize
                kernel(I)=LE3(I)
            end do
        case(10)
```

```
          do I=1,buffsize
               kernel(I)=HLE(I)
          end do
     case(11)
          do I=1,buffsize
               kernel(I)=VLE(I)
          end do
     case(12)
          do I=1,buffsize
               kernel(I)=MVE(I)
          end do
     case(13)
          do I=1,buffsize
               kernel(I)=MHE(I)
          end do
     case(15)
          print*,'Input filter coefficients in COLUMN MAJOR
order!'
          do I=1,buffsize
               read*,kernel(I)
          end do
     case(16)
          do I=1,buffsize
               kernel(I)=SVE(I)
          end do
     case(17)
          do I=1,buffsize
               kernel(I)=SHE(I)
          end do
     case(18)
          do I=1,buffsize
               kernel(I)=SHV(I)
          end do
     case(19)
          do I=1,buffsize
               kernel(I)=NGE(I)
          end do
     case(20)
          do I=1,buffsize
               kernel(I)=SGE(I)
          end do
     case(21)
          do I=1,buffsize
               kernel(I)=EGE(I)
          end do
     case(22)
          do I=1,buffsize
               kernel(I)=WGE(I)
          end do
     case(23)
          do I=1,buffsize
          kernel(I)=NEG(I)
      end do
  case(24)
      do I=1,buffsize
          kernel(I)=SEG(I)
      end do
  case(25)
      do I=1,buffsize
          kernel(I)=SWG(I)
      end do
  case(26)
      do I=1,buffsize
          kernel(I)=NWG(I)
      end do
  case(29)
      print*,'Input filter coefficients in COLUMN MAJOR
order!'
      do I=1,buffsize
          read*,kernel(I)
      end do
  case default
      kernelno=0
  end select
  return
  end

c************************************************************
  subroutine rgetfilter(kernelno,kernel,buffsize)

  integer*1 kernelno,buffsize
  real*4 kernel(*)
$INCLUDE:'realfilt.inc'

  select case(kernelno)
  case(4)
      do I=1,buffsize
          kernel(I)=LP1(I)
      end do
  case(5)
      do I=1,buffsize
          kernel(I)=LP2(I)
      end do
  case(6)
      do I=1,buffsize
          kernel(I)=LP3(I)
      end do
  case(27)
      do I=1,buffsize
          kernel(I)=USM(I)
      end do
  case(30)
```

```
        print*,'Input filter coefficients in COLUMN MAJOR
order!'
        do I=1,buffsize
            read*,kernel(I)
        end do
    case default
        kernelno=0
    end select
    return
    end

c************************************************************
    subroutine filterselect(key,arr1,iy)

    integer*1 kernelno,kerrow,kercol,iy(512,512)
    integer*1 buffsize,offcol,offrow
    integer*2 rowextent,rowstart,colextent,colstart,idisp
integer*4 key,arr1(2),irem

    call filtermenu(kernelno)
    select case(kernelno)
        case(1:11,14,16:28)
            kerrow=3
            kercol=3
        case(12)
            kerrow=3
            kercol=5
        case(13)
            kerrow=5
            kercol=3
        case(15,29,30)
            print*,'Input # of filter COLUMNS and ROWS'
            read*,kercol,kerrow
        case default
            kernelno=0
    end select
    if(kernelno.ne.0) then
    call params(kerrow,kercol,colextent,rowextent,
colstart,rowstart)
    offcol=int1(kercol/2)
    offrow=int1(kerrow/2)
    call getdispbuff(idisp)
    call setbusbuff(idisp,irem)
    print*,'Destination frame?'
    read*,idisp
    buffsize=kerrow*kercol
    select case(kernelno)
    case(1:3,7:13,16:26,29)
    call intconv(key,arr1,iy,kernelno,kerrow,kercol,offcol,
*offrow,rowstart,rowextent,colstart,colextent,idisp,irem,buf
    fsize)
case(4:6,27,30)
    call reconv(key,arr1,iy,kernelno,kerrow,kercol,offcol,

*offrow,rowstart,rowextent,colstart,colextent,idisp,irem,buf
fsize)
case(14)
    call sobel(key,arr1,iy,kernelno,kerrow,kercol,offcol,

*offrow,rowstart,rowextent,colstart,colextent,idisp,irem,buf
fsize)
case(15)
    call median(key,arr1,iy,kerrow,kercol,offcol,offrow,
*rowstart,rowextent,colstart,colextent,idisp,irem,buffsize)
case(28)
    call homo(key,arr1,iy,kerrow,kercol,
offcol,offrow,rowstart,
*       rowextent,colstart,colextent,idisp,irem,buffsize)
    end select
    endif
    return
    end

c************************************************************
    subroutine filtermenu(kernelno)

    integer*1 kernelno

    print*,'Input Kernel #'
    print*,'   1 - HP1    High Pass 1                16 - SVE
Shift an      *d Diff Vertical Edge'
    print*,'   2 - HP2    High Pass 2                17 - SHE
Shift an      *d Diff Horizon  Edge'
    print*,'   3 - HP3    High Pass 3                18 - SHV
Shift an      *d Diff Hor/Vert Edge'
    print*,'   4 - LP1    Low Pass 1                 19 - NGE
North Gr      *adient Edge'
    print*,'   5 - LP2    Low Pass 2                 20 - SGE
South Gr      *adient Edge'
    print*,'   6 - LP3    Low Pass 3                 21 - EGE
East Gra      *dient Edge'
    print*,'   7 - LE1    Laplacian Edge 1           22 - WGE
West Gra      *dient Edge'
    print*,'   8 - LE2    Laplacian Edge 2           23 - NEG
North/Ea      *st Gradient Edge'
    print*,'   9 - LE3    Laplacian Edge 3           24 - SEG
South/Ea      *st Gradient Edge'
    print*,'  10 - HLE    Horizon  Line Enhance      25 - SWG
South/We      *st Gradient Edge'
```

```
      print*,'  11 - VLE    Vertical Line Enhance    26 - NWG
North/We        *st Gradient Edge'
      print*,'  12 - MVE    Matched Vertical Edge    27 - USM
Unsharp         *Masking'
      print*,'  13 - MHE    Matched Horizon Edge     28 - HMO
Homomorp        *hic Filter'
      print*,'  14 - SOB    Sobel Edge Detection     29 - USR1
USER DEF        *INED - INTEGER'
      print*,'  15 - MED    Median Filter            30 - USR2
USER DEF        *INED - REAL'
      read*,kernelno
      return
      end

c************************************************************
      subroutine median(key,arr1,iy,
       * kerrow,kercol,offcol,offrow,rowstart,rowextent,
      *  colstart,colextent,idisp,irem,buffsize)

      integer*2 idisp,error,time
      integer*4 key,ier,arr1(2),bytes,irem,ndx
      integer*2 pix,colextent,colstart,rowextent,rowstart
      integer*2 buff[ALLOCATABLE](:),iret,ier1
      integer*1 iy(512,512),kerrow,kercol,offcol
      integer*1 buffsize,buffmed,offrow,result(512)

      ALLOCATE (buff(buffsize),stat=error)
      buffmed=int1(buffsize/2)+1
      call getxtd(2,arr1,4,-1,ihndl,key1,kb,iret,ier1)
      if(ier1.ne.0) then
          print*,'getxtd error=',ier1
          goto 99
      end if
      print*,'This will take a few minutes'
      call gettim(ihour,imin,isec,ihn)
      call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
      if(ier.ne.0) print *,'ia2axtd error = ',ier
      call setbusbuff(idisp,irem)
      do I=rowstart,rowextent
      do J=colstart,colextent
          pix=1
          do M=0,kerrow-1
              do N=0,kercol-1
              buff(pix)=iy(J+N,I+M)
              if(buff(pix).lt.0)buff(pix)=256+buff(pix)
              pix=pix+1
              end do
          end do
      call sort(buff,buffsize)
      result(J+offcol)=buff(buffmed)
      end do
```

```
    ndx=(512*(I+offrow-1)+offcol)
    call a2axtd(1,128,4,locfar(result),
key+irem+ndx,bytes,ier)
    if(ier.ne.0)print*,'a2axtd error =',ier
    end do
    call display(idisp)
    call gettim(nhour,nmin,nsec,nhn)
    DEALLOCATE (buff, Stat=error)
    time=(nmin-imin)*60 + (nsec-isec)
    print*,time,' secs'
99      call relxtd(ihndl,iret,ier)
    if(ier.ne.0) print *,'relxtd error = ',ier,iret
    return
    end
c************************************************************
    subroutine params(kerrow,kercol,colextent,rowextent,
      *  colstart,rowstart)

    integer*1 kerrow,kercol
    integer*2 rowstart,rowstop,colextent,rowextent
    integer*2 colstart,colstop
    character*4 ians

    print*,'Default window is 512x512 image'
    print*,'                 Accept Default ? (y/n)'
    read(*,'(A)') ians
1       if((ians.eq.'n').or.(ians.eq.'N')) then
        print*,'Input COLUMN Start and Stop'
        read*,colstart,colstop
        print*,'Input ROW Start and Stop'
        read*,rowstart,rowstop
      if((colstop.le.colstart).or.(rowstop.le.rowstart))
goto 1
        colextent=(colstop-colstart)-kercol+1
        rowextent=(rowstop-rowstart)-kerrow+1
    else
        colextent=512-kercol+1
        rowextent=512-kerrow+1
        colstart=1
        rowstart=1
    endif
    return
    end

c************************************************************
    subroutine sort(ibuff,imax)
    integer*1 switch,limit,imax
    integer*2 ibuff(*),itemp

    limit  = imax
    do while( limit .ne. 0 )
```

```
          switch = 0
          do I = 1, limit-1
          if(ibuff(I).gt.ibuff(I+1)) then
              itemp= ibuff(i)
              ibuff(i) = ibuff(I+1)
              ibuff(i+1) = itemp
              switch = I
          end if
          end do
          limit = switch
      end do
      return
      end

c************************************************************
      subroutine sobel(key,arr1,iy,
       *   kernelno,kerrow,kercol,offcol,offrow,rowstart,
      *   rowextent,colstart,colextent,idisp,irem,buffsize)

      integer*2 error,idisp,time
      integer*4 key,ier,arr1(2),bytes,irem,ndx
      integer*2 rowextent,rowstart,colextent,colstart
      integer*1 buffsize,kernelno,iy(512,512)
      integer*1 kerrow,kercol,offcol,offrow,pix,result(512)
real*4 above,below,AEI,BEH,DEF,CEG,maxdif
      real*4 buff[ALLOCATABLE](:)

      ALLOCATE (buff(buffsize),stat=error)
      print*,'This will take a few minutes'
      call gettim(ihour,imin,isec,ihn)
      call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
      if(ier.ne.0)print*,'a2axtd error(iy) =',ier
      call setbusbuff(idisp,irem)
      do I=rowstart,rowextent
      do J=colstart,colextent
          pix=1
          rsum=0.0
          do M=0,kercol-1
          do N=0,kerrow-1
              buff(pix)=iy(J+M,I+N)
              if(buff(pix).lt.0.0)buff(pix)=256.0+buff(pix)
              if(buff(pix).eq.0.0)buff(pix)=1
              pix=pix+1
          end do
      end do
      above=real(buff(4)+buff(7)+buff(8))/3
      below=real(buff(2)+buff(3)+buff(6))/3
      AEI=abs(below-above)
      above=real(buff(1)+buff(4)+buff(7))/3
      below=real(buff(3)+buff(6)+buff(9))/3
      BEH=abs(below-above)
```

```
    above=real(buff(7)+buff(8)+buff(9))/3
    below=real(buff(1)+buff(2)+buff(3))/3
    DEF=abs(below-above)
    above=real(buff(6)+buff(8)+buff(9))/3
    below=real(buff(1)+buff(2)+buff(4))/3
    CEG=abs(below-above)
    maxdif=max(AEI,BEH)
    maxdif=max(CEG,maxdif)
    maxdif=max(DEF,maxdif)
    if(maxdif.gt.255.0)maxdif=255.0
    if(maxdif.lt.0.0)maxdif=0.0
    result(J+offcol)=maxdif
    end do
    ndx=(512*(I+offrow-1)+offcol)
    call a2axtd(1,128,4,locfar(result),
key+irem+ndx,bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(result) =',ier
    end do
    call gettim(nhour,nmin,nsec,nhn)
    call display(idisp)
    time=(nmin-imin)*60 + (nsec-isec)
    print*,time,' secs'
    DEALLOCATE (buff, Stat=error)
    return
    end

c************************************************************
subroutine homo(key,arr1,iy,kerrow,kercol,offcol,offrow,
*rowstart,rowextent,colstart,colextent,idisp,irem,buffsize)
    integer*2 error,error1,idisp,time,kernel[ALLOCATABLE](:)
    integer*4 key,ier,arr1(2),bytes,irem,ndx
    integer*2 rowextent,rowstart,colextent,colstart
    integer*1 buffsize,kernelno,iy(512,512)
    integer*1 kerrow,kercol,offcol,offrow,pix,result(512)
real*4 buff[ALLOCATABLE](:)

    ALLOCATE (buff(buffsize),stat=error)
    ALLOCATE (kernel(buffsize),stat=error1)
    kernelno=1
    call igetfilter(kernelno,kernel,buffsize)
    print*,'This will take a few minutes'
    call gettim(ihour,imin,isec,ihn)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(iy) =',ier
    call setbusbuff(idisp,irem)
    do I=rowstart,rowextent
    do J=colstart,colextent
        pix=1
        rsum=0.0
        do M=0,kercol-1
        do N=0,kerrow-1
```

```
            buff(pix)=iy(J+M,I+N)
            if(buff(pix).lt.0.0)buff(pix)=256.0+buff(pix)
            if(buff(pix).eq.0)buff(pix)=1.0
            rsum=rsum+(log(buff(pix))*kernel(pix))
            pix=pix+1
        end do
    end do
    result(J+offcol)=exp(rsum)
    end do
    ndx=(512*(I+offrow-1)+offcol)
    call a2axtd(1,128,4,locfar(result),
key+irem+ndx,bytes,ier)
    if(ier.ne.0)print*,'a2axtd error(result) =',ier
    end do
    call gettim(nhour,nmin,nsec,nhn)
    call display(idisp)
    time=(nmin-imin)*60 + (nsec-isec)
    print*,time,' secs'
    DEALLOCATE (buff, Stat=error)
    DEALLOCATE (kernel, Stat=error1)
    return
    end

c*************************************************************
    subroutine stats(key,arr1,iy)
c       Calculates histogram, mean, std

    integer*2 idisp,error,pix,colstart
    integer*2 rowstart,colextent,rowextent
    integer*4 key,ier,arr1(2),bytes,irem,total
    integer*4 hist[allocatable](:),sum
    integer*1 iy(512,512),itemp
    real*4 mean,var,std
    character*64 fileout
    character*4  ans

    call getdispbuff(idisp)
    call setbusbuff(idisp,irem)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0) print *,'ia2axtd error = ',ier
    allocate (hist(0:255), Stat=error)
    do I=0,255
        hist(I)=0
    end do
    itemp=1
    call params(itemp,itemp,colextent,
rowextent,colstart,rowstart)
    total=(rowextent-rowstart+1)*(colextent-colstart+1)
    sum=0.0
    var=0.0
    do I=rowstart,rowextent
```

```
        do J=colstart,colextent
        pix=iy(J,I)
        if(pix.lt.0) pix=256+pix
        hist(pix)=hist(pix)+1
        sum=sum+int4(pix)
        var=var+ real(pix)*real(pix)
        end do
    end do
    mean=real(sum)/real(total)
    std=sqrt((var -
real(sum)*real(sum)/real(total))/real(total-1))
    print*,'Sum=',sum
    print*,'Pixels=',total
    print*,'Mean=',mean
    print*,'Std Deviation=',std
    print*,'Do you wish to save stats and histogram to file?
            (y,n)'
    read(*,'(A)') ans
    if((ans.eq.'y').or.(ans.eq.'Y')) then
    print*,'Input filename to store HISTGRAM data'
    read(*,'(A)') fileout
    open(3,file=fileout,form='formatted')
    do I=0,255
        write(3,'(I8)') hist(I)
    end do
        write(3,'(I10)') sum
        write(3,'(I8)') total
        write(3,'(F8.4)') mean
        write(3,'(F8.4)') std
    close(3)
    endif
    deallocate (hist,stat=error)
    return
    end

c*************************************************************
    subroutine expandaga(key,iy)
c       expand AGA image by 4

    integer*2 idisp,colstart,L,rowstart
    integer*4 key,ier,bytes,irem,total
    integer*1 iy(512,512),itemp(512)

    call getdispbuff(idisp)
    call setbusbuff(idisp,irem)
    print*,'Input COLUMN Start of AGA Image (1-12)'
    read*,colstart
    print*,'Input ROW Start of AGA Image (1-12)'
    read*,rowstart
    print*,'Destination Frame?'
    read*,idisp
    call a2axtd(1,65536,4,key+irem,locfar(iy),bytes,ier)
if(ier.ne.0) print *,'a2axtd error = ',ier
    call setbusbuff(idisp,irem)
    total=irem
    do I=185+rowstart,185+rowstart+127
        L=0
        do J=185+colstart,185+colstart+127
        itemp(L+1)=iy(J,I)
        itemp(L+2)=iy(J,I)
        itemp(L+3)=iy(J,I)
        itemp(L+4)=iy(J,I)
        L=L+4
        end do

    call a2axtd(1,128,4,locfar(itemp),key+total,bytes,ier)
if(ier.ne.0) print *,'a2axtd error(1) = ',ier
    total=total+int4(512)
    call a2axtd(1,128,4,locfar(itemp),key+total,bytes,ier)
if(ier.ne.0) print *,'a2axtd error(2) = ',ier
    total=total+int4(512)
    call a2axtd(1,128,4,locfar(itemp),key+total,bytes,ier)
if(ier.ne.0) print *,'a2axtd error(3) = ',ier
    total=total+int4(512)
    call a2axtd(1,128,4,locfar(itemp),key+total,bytes,ier)
if(ier.ne.0) print *,'a2axtd error(4) = ',ier
    total=total+int4(512)
    end do
    call display(idisp)
    return
    end
c************************************************************
    subroutine compress(key,arr1,iy)
c       Compresses an IRSTD file by a factor of 5
    integer*2 idisp
    integer*4 key,bytes,ier
    integer*4 arr1(2),irem
    integer*1 iy(512,512),compact(512)

    call getdispbuff(idisp)
    call setbusbuff(idisp,irem)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    print*,'Destination Frame ?'
    read*,idisp
    call setbusbuff(idisp,irem)
    K=0
    do I=1,450,5
    do J=1,512
        compact(J)=iy(J,I)
    end do
    ndx=K*512
    K=K+1
    call a2axtd(1,128,4,locfar(compact),
key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    end do
    call display(idisp)
    return
    end
c************************************************************
    subroutine expand(key,arr1,iy)
c       Expand an IRSTD image by a factor of 5

    integer*2 idisp
    integer*4 key,bytes,ier
    integer*4 irem,arr1(2)
    integer*1 iy(512,512),temp(512)

    call getdispbuff(idisp)
    call setbusbuff(idisp,irem)
    call a2axtd(2,arr1,4,key+irem,locfar(iy),bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    print*,'Destination Frame ?'
    read*,idisp
    call setbusbuff(idisp,irem)
    ndx=0
    do I=1,90
    do J=1,512
        temp(J)=iy(J,I)
    end do
    call a2axtd(1,128,4,locfar(temp),key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    ndx=ndx+512
    call a2axtd(1,128,4,locfar(temp),key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    ndx=ndx+512
    call a2axtd(1,128,4,locfar(temp),key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    ndx=ndx+512
    call a2axtd(1,128,4,locfar(temp),key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    ndx=ndx+512
    call a2axtd(1,128,4,locfar(temp),key+irem+ndx,bytes,ier)
    if(ier.ne.0) print *,'a2axtd error= ',ier
    ndx=ndx+512
    end do
    call display(idisp)
    return
    end
c************************************************************
    subroutine adjust(key,arr1,iy)
c       Supplies menu to expand/compress AGA/IRSTD files
```

```
integer*4 key,arr1(2)
integer*1 iy(512,512),ians

print*,'Enter   1  to expand AGA image'
print*,'        2  to compress IRSTD image'
print*,'        3  to expand IRSTD image'
read*,ians
select case(ians)
case(1)
call expandaga(key,iy)
case(2)
call compress(key,arr1,iy)
case(3)
call expand(key,arr1,iy)
end select
return
end
```

INTERFCE.INC

```
c*************************************************************
c   This file contains to appropriate 'interface' statements
c   for the x-arRAY subroutines used in the processing c
program
c*************************************************************
    interface to subroutine a2axtd (d,n,lmnt,key[value],
*  key1[value],bytes,ier)
    integer*4 d,n,lmnt,key,key1,bytes,ier
    end
c*************************************************************
    interface to subroutine getxtd (d,n,lmnt,mthd,ihndl,
*  key,kb,iret,ier)
    integer*4 d,n,lmnt,key,kb,mthd,ihndl
    integer*2 ier,iret
    end
c*************************************************************
 interface to subroutine ssmrnm(D,N,key[value],xscale,ier)
    integer*4 d,n,ier,key
    real*4 xscale
    end
c*************************************************************
    interface to subroutine ismlum(D,N,key[value],scl,ier)
integer*4 d,n,ier,key
    real*4 scl
    end
c*************************************************************
interface to subroutine implum(D,N,key[value],key1[value],
  *  key2[value],ier)
    integer*4 d,n,ier,key,key1,key2
    end
c*************************************************************
interface to subroutine ilnlum(D,N,C1,key[value],C2,      *
key1[value],K,key2[value],ier)
    integer*4 d,n,ier,key,key1,key2,k
    real*4 c1,c2
    end
```

FILTER DEFINITIONS

```
c       INTFILT.INC

      integer*1  HP1(9),HP2(9),HP3(9),MVE(15)
      integer*1  SVE(9),SHE(9),SHV(9),MHE(15)
      integer*1  LE1(9),LE2(9),LE3(9),HLE(9),VLE(9)
      integer*1  NGE(9),SGE(9),EGE(9),WGE(9)
      integer*1  NEG(9),SEG(9),SWG(9),NWG(9)

c Filter data below is in column major order
      data HP1/ -1,-1,-1, -1,9,-1, -1,-1,-1/
      data HP2/ 0,-1,0, -1,5,-1, 0,-1,0/
      data HP3/ 1,-2,1, -2,5,-2, 1,-2,1/
      data SVE/ 0,-1,0, 0,1,0, 0,0,0/
      data SHE/ 0,0,0, -1,1,0, 0,0,0/
      data SHV/ -1,0,0, 0,1,0, 0,0,0/
      data LE1/ 0,1,0, 1,-4,1, 0,1,0/
      data LE2/ -1,-1,-1, -1,8,-1, -1,-1,-1/
      data LE3/ 1,-2,1, -2,4,-2, 1,-2,1/
      data NGE/ 1,1,-1, 1,-2,-1, 1,1,-1/
      data SGE/ -1,1,1, -1,-2,1, -1,1,1/
      data EGE/ -1,-1,-1, 1,-2,1, 1,1,1/
      data WGE/ 1,1,1, 1,-2,1, -1,-1,-1/
      data NEG/ 1,-1,-1, 1,-2,-1, 1,1,1/
      data SEG/ -1,-1,1, -1,-2,1, 1,1,1/
      data SWG/ 1,1,1, -1,-2,1, -1,-1,1/
      data NWG/ 1,1,-1, 1,-2,-1, 1,1,-1/
      data MVE/ -1,-1,-1,-1,-1, 0,0,0,0,0, 1,1,1,1,1/
      data MHE/ -1,0,1, -1,0,1, -1,0,1, -1,0,1, -1,0,1/
      data VLE/ -1,-1,-1, 2,2,2, -1,-1,-1/
      data HLE/ -1,2,-1, -1,2,-1, -1,2,-1/


c   REALFILT.for

      real*4  LP1(9),LP2(9),LP3(9),USM(9)

c Filters data below is in column major order
      data LP1/ .111111111,.111111111,.111111111,
       *            .111111111,.111111111,.111111111,
       *            .111111111,.111111111,.111111111/
      data LP2/ .1,.1,.1, .1,.2,.1, .1,.1,.1/
      data LP3/ .0625,.125,.0625, .125,.25,.125,
       *            .0625,.125,.0625/
      data USM/ -.125,-.125,-.125, -.125,1,-.125,
       *            -.125,-.125,-.125/
```

APPENDIX B

A number of useful utility programs were developed for converting AGA image files to a format compatible with the IRIS Tutor processing programs. A brief description of each function is provided.

- AGA2IMG - This utility converts a 20,446 byte AGA image file into a 262,656 byte IRIS image file by centering the 140x140 AGA data area in a 512x512 array, then adding an appropriate file header. An AGA image file has a 846 byte header, followed by 19,600 bytes of image data. An IRSTD image file contains a 512 byte header, followed by 262,144 bytes of data. The conversion program discards the 846 byte AGA header since the IRIS file format does not incorporate similar header fields in its 512 byte header. The 19,600 byte data area of the AGA file is then read into a blank 512x512 array, so that AGA data is centered within the 512x512 array. The resulting data array (19,600 bytes of good data and 242,544 bytes of blanks) is written to a new file along with a 512 byte header, and an IRIS compatible image file is created.

- AGA2IMG1 - Same basic utility as in AGA2IMG, however the image is automatically expanded to 512x512 by disregarding 6 lines on all sides of the image.

- AGAHDR - This file extracts the header information from an original AGA image file, and writes the information to a user specified location. This routine can be command-line-accessed by input of the name of the file to be processed followed by the filename for the resulting header information.

APPENDIX C

The following assembly language code replaces the final copy routine in Bernier's 'LOADUP' program [Ref. 4] to enable the load of images to consecutive frame buffers, thus eliminating the need to halt the acquisition program every time a file of interest is created. By first determining which frame buffer is currently being displayed, and then executing a copy to the next sequential frame buffer, up to 14 consecutive rotations can be acquired without halting the program (the program sequences from buffer 15 to buffer 2 so as not to disturb the sorting process). The determination of the correct display buffer and the actual copy are performed using the DT-2861 control/status registers rather than executing a word-by-word copy as was done previously. One consequence of this implementation is the loading of an extra 90 lines of data, but this has not shown any adverse effects. The program which fully implements this feature is entitled 'MULTLOAD'.

```
Subroutine buffcpy.pm
by J.C. Heiss
MASM 5.1
19 May 1992

; ***********  BUFFCPY ROUTINE

curfrm: mov dx, YPAN
    in  ax, dx        ;get current frame number
    shr ax, 12
    cmp ax, 000Fh       ;check to see if at 15 yet
    jl  cont
    mov ax, 0001h       ;if at 15, go to frame 1

cont:inc ax             ;advance to next sequential frame
    push ax             ;save new frame number
    mov dx, INCSR2      ;set register for FEEDBACK frame 0
    bts ax, 7           ;BSEL set, BUFFSEL=new frame number
    out dx, ax

    mov dx, INCSR1      ;set register for ALU=1010
    mov ax, 00A0h
    shl ax, 8           ;BUSY set, ALUM set
    add ax, 0088h
    out dx, ax          ;results in ALU function F=B

    pop ax
    shl ax, 12          ;sets display buffer to next frame
    mov dx, YPAN
    out dx, ax

skip4:
    mov ax, BUS01        ;restore buffer 0 at base address
0A00000h
    mov dx, OUTCSR
    out dx, ax

busy2:mov dx, INCSR1   poll BUSY bit til operation completed
    in  ax, dx
    bt  ax, 7
    jc  short busy2

;Perform absolute 16 bit jump (in a 16 bit segment)
    JMP32S nextfrm

;End of program
quit:mov ax, 21h
    mov PINTFRAME.VMINT,eax
    mov edx, OFFSET endmsg     ;display end of prog message

    mov ah, 9
    mov ebx, OFFSET PINTFRAME
    VM86CALL

    pop ecx           ;clean up stack
    popad             ;restore all 32-bit registers from stack
    BACK2DOS
USER    ENDP
    PROT_CODE_END
```

LIST OF REFERENCES

1. Klass, Phillip J., "Interest Rises in Infrared Search, Track", *Aviation Week and Space Technology*, August 30, 1982, pp. 68-71.

2. Itakura, Y., Tsutsumi, S. and Takagi, T., "Statistical Properties of the Background Noise for the Atmospheric Windows in the Intermediate Infrared Region," *Infrared Physics*, v. 14, pp. 17-29, 1974.

3. Cooper, A.W., NACIT Proposal for Research for period 1 October 1991 to 30 September 1992, Naval Sea Systems Command, PMS-421.

4. Bernier, J.D., "Real Time Imaging and Infrared Background Analysis using the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System", Masters Thesis, Naval PostGraduate School, September 1991.

5. Engel, R.C., "A PC Based Imaging System for the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System", Master's thesis, Naval Postgraduate School, Monterey, CA, September 1989.

6. User Manual for the DT-2861 and DT-2862 Arithmetic Frame Grabbers for the IBM Personal Computer AT, 5th ed., Data Translation, Inc., Marlboro, MA, 1989.

7. Cooper, A.W., W.J. Lentz, M.J. Baca and J.D. Bernier, "Image Display and Background Analysis with the Naval Postgraduate School Infrared Search and Track System", *SPIE Proceedings*, Vol. 1486, 1991.

8. Ayers, Gary, R., "Calibration and Initialization of the NPS Modified Infrared Search and Target Designation (IRSTD) System", Master's thesis, Naval Postgraduate School, Monterey CA, December 1987.

9. McKaig, T.R., "Thermal Imaging with AGA Thermovision 780", Master's thesis, Naval Postgraduate School, Monterey, CA, December, 1897.

10. AGA Thermovision 780 Operating Manual, AGA Infrared Systems, 1980.

11. Bayar, S., "Statistical Analysis of Background IR Emission in the 3-5.6 μm and 8-14 μm Regions", Master's thesis, Naval Postgraduate School, Monterey, CA, December, 1989.

12. Paul and Meg Noah, VGAIPS User's Manual, Disk file accompanying the VGAIPS program.

13. IRIS Tutor User Manual, 3rd ed., Data Translation Inc., Marlboro, MA, 1987.

14. X-arRAY Extended Memory from Fortran Reference Manual, Davis Associates, Inc., West Newton, MA, 1992.

15. PCTOOLS User's Manual, ONTAR Corporation, Brookline, MA.

16. Lindley, Craig A., *Practical Image Processing in C*, John Wiley and Sons, Inc., 1991.

17. Schalkoff, Robert J., *Digital Image Processing and Computer Vision*, John Wiley & Sons, Inc., 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Chairman, Code PH Department of Physics Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Professor A.W. Cooper, Code PH/Cr Department of Physics Naval Postgraduate School Monterey, CA 93943-5000	2
6.	Naval Sea Systems Command Attn: J.E. Misanen, PMS-421 Washington, DC 20362-5101	1
7.	Naval Surface Warfare Center, White Oak Detachment Attn: S.K. Petropoulous, Code R42 Silver Springs, MD 20903-5000	1
8.	Naval Surface Warfare Center Attn: M.W. Zurasky, Code 6552 Dahlgren, VA 22448-5000	1
9.	Robert J.L. Corriveau Director, Electro-Optics Division Defense Research Establishment, Valcartier 2459 Pie XI Blvd., North (P.O. Box 8800) Courcelette, Quebec, Canada, G0A 1R0	1

10.	Dr. Loris G. Gregoris Spar Aerospace Limited Director, Electro-Optical Technology 1235 Ormont Drive Weston, Ontario, Canada, M9L 2W6	1
11.	Office of Naval Technology Attn: James Buss, Code 2141 800 North Quincy St. Arlington, VA 22217-5000	1
12.	LT John C. Heiss, USN 2606 Liter Ct. Ellicott City, MD 21043	1